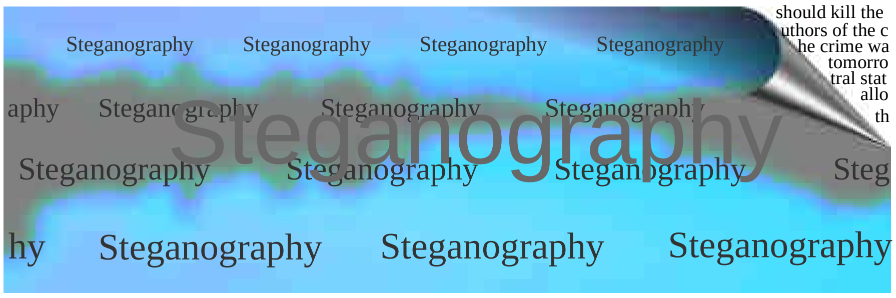
The “Politehnica” University of Timisoara

Faculty of Automatics and Computers

Department of Computer Science and Software Engineering



An Analysis of Steganographic Techniques

by Richard Popa

Scientific advisers:

Prof. Andrew S. Tanenbaum, Ph.D.

Christoph Hänle

Prof. Ioan Jurca, Ph.D.

vrije Universiteit amsterdam

UPT universitatea politehnica timisoara

1998

Abstract

Modern computer networks make it possible to distribute documents quickly and economically. This is because of the decreasing cost of the equipment needed to copy, to print, to process the information. The widespread adoption of electronic distribution of copyrighted material is accompanied by illicit copying and illicit distribution. This is why people think about how to protect their work, how to prevent such illicit activities and how to trace the distribution of a document. Steganography is a technique that allows users to hide a message in another message; using steganography and digital watermarks it is possible to hide copyright information, such the ID of the author, the date of creation, etc. In this paper we take a look to the existent methods used for embedding information into a large variety of document types: document images, audio files, image files, binary files.

CONTENTS

Introduction

The appearance of the Internet is considered to be one of the major events of the last years; information become available on-line, all users who have a computer can easily connect to the Internet and search for the information they want to find. The result is that everybody can read the latest news on-line and also consult digital libraries, read about firms, universities, cultural events, exhibitions, etc. but also the companies can sell their products through the Internet, using electronic commerce.

In the last few years there was a rapidly growing interest in ways to hide information in other information. The fact that an unlimited number of perfect copies can be illegally produced led people to study ways of embedding hidden copyright information and serial numbers in audio and video data; concern that privacy could be eroded led to anonymous remailers, techniques for making mobile computers harder for the third party to trace; the restrictions of some governments concerning the availability of encryption services motivated people to study and find methods for communicating secretly. Those are new techniques that the scientists are currently developing and who are in permanent improvement.

Cryptography can be defined as secret writing. The word *cryptography* comes from the Greek words *κρυπτο* (secret, hidden) and *γραφη* (writing)[ChKau95] The art is mangling information into apparent unintelligibility in a manner allowing a secret method of unmangling. The basic service that cryptography offers is the ability of transmitting information between persons in a way that prevents a third party from reading it. Cryptography can also provide *authentication* (verifying the identity of someone or something). Cryptographic systems usually involve both an algorithm and a secret key. The reason for having a secret key is that keeping the algorithm secret is very difficult and it is also difficult to quickly explain a new algorithm to a person. We can say that with a good cryptographic scheme it is OK for everyone to know the algorithm (including the possible attackers), because the message cannot be unmangled without the knowing of the key and also public review by many independent cryptographers helps to find security flaws more reliably. In fact this is *the Kerchoff's principle*: the security of the algorithm resides in the secret key; without the secret key, any attack has very little chance to succeed.

Encryption is the process of disguising a message in such a way as to hide its substance. This process requires a key for the encoding process and a key (which may be the same key) for the decoding process. Using such a technique, two parties that share a pair of keys can easily communicate in a secure way across an insecure environment. A third party cannot see the content of the message if he does not have the decrypting key. The attacker can truncate, modify, replay, absorb, analyze the message, but he can not remove the encryption (see Table 1).

Digital signatures are used as a proof of authorship of the contents of a document. Two different techniques are used for signing an object. With the first method the sender attaches a message digest to the original message and the signed message obtained is transmitted through the communication environment. The receiver recalculates the digest and compares it to the one received. With the second method the two parties use *public-key algorithms*. Public-key algorithms use a pair of keys for each party involved in the communication: a *secret key*, known only to the owner, and a *public key*, known to everybody. The sender encrypts the document using his secret key and sends it to the receiver, which decrypts the message using the public key of the sender. In both alternatives, if the decryption process is successful then the message is authentic. The digital signatures may be detached, but even if there is only a small change to the message, the signature attached is no longer valid.

	Confidentiality	**Integrity**	**Unremovability**
Encryption	Yes	No	Yes
Digital Signatures	No	Yes	No
Steganography	Yes/No	Yes/No	Yes

Table 1. Comparison between encryption, digital signatures and steganography.

The word *steganography* comes from the Greek *στεγανω* (covered writing). [DaKah96] Steganography is an ancient art of embedding private messages in seemingly innocuous messages in such a way that prevents the detection of the secret messages by a third party. In other words, steganography means establishing *covert channels*. A covert channel is a secret communication channel used for transmitting information. As shown in Figure 1, two general directions can be distinguished within steganography: *protection against detection* and *protection against removal*. [ChCac98] Protection against detection is achieved using schemes that do not modify in a visible way the original unmarked object; the modifications are not visible by the humans or by the computers. Protection against removal supposes that the scheme should be robust to common attacks; it is impossible to remove the hidden data without degrading the object's quality and rendering it useless.

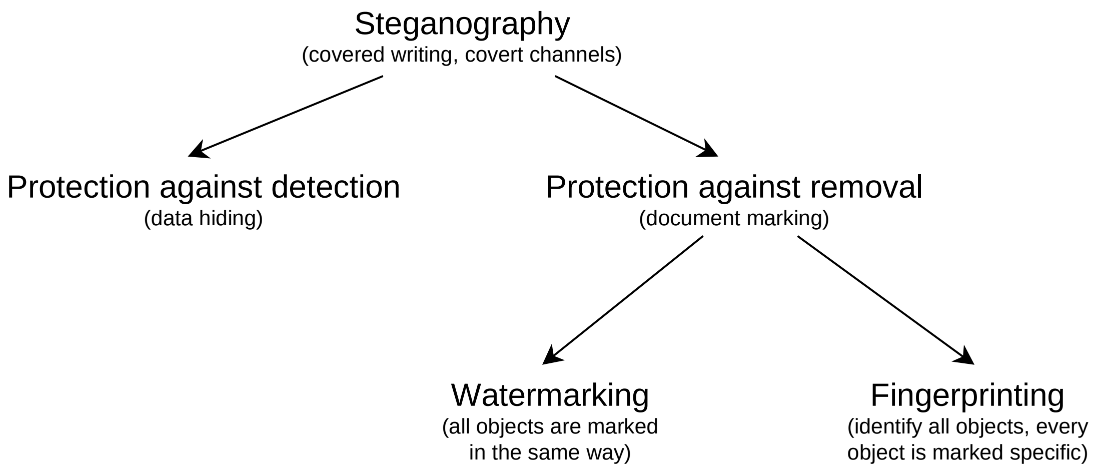

Steganography
(covered writing, covert channels)
Protection against detection
(data hiding)
Protection against removal
(document marking)
Watermarking
(all objects are marked in the same way)
Fingerprinting
(identify all objects, every object is marked specific)

Figure 1. Directions within steganography.

The first approach, protection against detection, can be illustrated using the *Prisoner's Problem*, formulated by Simmons in 1983. [RoAnd98] According to this scenario, Alice and

Bob[1] are in jail, and they wish to hatch an escape plan. All their communications pass through Willie, the warden. If Willie detects any encrypted or suspicious message he will frustrate their plan. So Alice and Bob must find some way of hiding their ciphertext in an innocuously looking covertext. As in cryptography, we assume that the security should depend only on a secret key that Alice and Bob have somehow managed to share. Supposing that the two prisoners communicate by sending pictures, then the general model is that Alice can add an extra percent of noise, without affecting the quality of the image. Willie can do the same thing too, but the quantity of the noise is limited, because of the degradation introduced in the picture. By using a keystream to select the bits she will change, Alice can stop Willie from finding out where she has put the information. So, if Willie wants to add extra noise he must add it to all possible locations in the image and the image will be distorted.

The second approach, protection against removal, is used for document marking, for embedding information about the author or for embedding a serial number; in other words, copyright information. In this case, the goal is protection against removal; the watermark might or might not be made visible to the user using a watermarking reader tool. There are two different techniques for document marking: *watermarking* and *fingerprinting*. Watermarking is the process of embedding marks in digital documents (sounds, images, binaries, etc.) exactly like the watermarks used for example for marking a banknote. Fingerprinting is the process of embedding a serial number into every copy of an object. This serial number can be used to detect the break of a license agreement. In both cases, the information is supposed to be invisible, but it should be very difficult to remove it. The difference between the two processes is that in the former process the objects are all marked the same way, but in the latter process every copy has a different serial number embedded.

Because the only difference between the watermarking and fingerprinting is the type of the information embedded, not the techniques used for inserting or their properties, we will use the term watermarking for document marking.

There is a difference between the two approaches discussed above: in the Prisoners' Problem a successful attack consists in finding out that a given object is marked, finding out that the two prisoners communicate. In the latter approach, watermarking, a successful attack consists in detecting the watermark and rendering it useless, in other words to detect and to extract the watermark. As a result, in some cases the data inserted should be robust to attacks.

If in one application we want to achieve confidentiality, than we have two alternatives: encryption or steganographic techniques for protection against detection (see Table 1 and

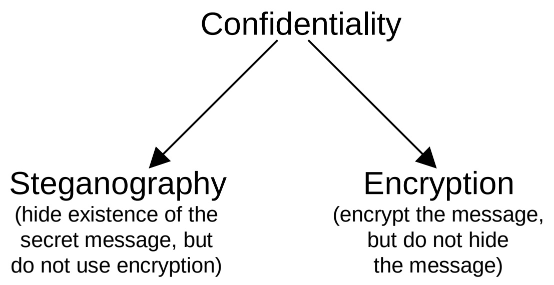

Confidentiality
Steganography
(hide existence of the secret message, but do not use encryption)
Encryption
(encrypt the message, but do not hide the message)

Figure 2. Achieving confidentiality.

[1] Communication protocols usually involve two fictional characters, named Alice and Bob. The standard convention is to name the participants either alphabetically (Carol and Dave often succeed Alice and Bob), or with a name whose first letter matches the first letter of their role, such as Willie the warden.

Figure 2). If we use encryption, only the owner of a key (that is the receiver) can read the secret message, but anybody can see that the two parties communicate secretly. If we use steganography, the very existence of the secret message is hidden; it does not use encryption.

Protecting the object's contents is another goal in many applications (see Figure 3). What the author wants is to *seal* his document or to *trace* it. Sealing the documents is achieved using digital signatures. This is equivalent to read-only protection; even the slightest modifications to the original object are detected. Unlike digital signatures, the watermarks used for tracing documents are spread all over the covertext, while the length of the document is unchanged. If the document is changed in some way then the watermark will remain as long as the original marked data remains inside the document. To put it in another way, the signatures answer to the following question: "is the document changed?" and the watermarks answer to the question: "was the document ever signed?"

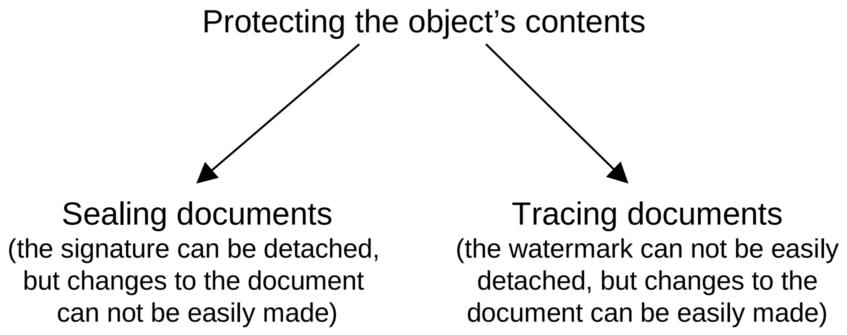

Protecting the object's contents
Sealing documents
(the signature can be detached, but changes to the document can not be easily made)
Tracing documents
(the watermark can not be easily detached, but changes to the document can be easily made)

Figure 3. Protecting the documents.

The parameters of the watermarks are *bit rate, hardware requirements* and *universality*. The bit rate refers to the quantity of the information a watermark can encode in a signal. An equivalent term is *channel capacity*. Concerning the hardware requirements, in some applications the encoding/decoding cost is important. In some cases this must be done in real time. Universality refers to finding algorithms that can be applied to more than one file type, for example to text, audio, image and video, resulting in the implementation of those algorithms on common hardware.

Information may be embedded not only to achieve secrecy, but also to combine audio with image; for example a short audio clip may associate a train whistle with an image of a locomotive. A slightly different application is the automatic monitoring of radio advertisements, where it would be convenient to have an automated system to verify that adverts are played as contracted. Another application is to transmit an ID during the radio transmission and if the receiver has a special decoder it can display the name of the radio station.

There are several steganographic techniques. We know that within the network the packets are transmitted using some rules. A network packet usually has headers, user data and trailers at the end. The sender adds both types of data to the user data and the receiver strips it off, its purpose is for example to correctly route the packet inside the network or to perform error detection or recovery. The only information that is passed to the application is the user data. Covert channels can be established using the control data, timing properties of transmission or of the user data. In the first two cases, control data and timing, there is very difficult or almost impossible to prove later the existence of covert channels, because the information is stripped off at the receiver. But if the information is hidden in the user data, it remains on the disk until the user deletes it.

This paper is organized as follows. In the first chapter we will present a short history of steganography, in chapter two we will discuss some techniques for data hiding in control data and timing properties of transmission. In chapter 3 we will present techniques for embedding information in the user data of different types: document images, sound files, image files and binaries. Chapter four will present some consequences of using such techniques.

Chapter 1. A Short History of Steganography

Throughout history, the people have tried to find methods to hide information. In fact, they have used a multitude of such techniques and variations. David Kahn provides a very interesting history in the book named *The Codebreakers*. There is also Bruce Norman who recounts numerous tales of cryptography and steganography during wars in the book *Secret Warfare: The Battle of Codes and Ciphers*. [DaKah96][NeJoh98]

One of the first documents describing steganography is from the Histories of Herodotus, the father of history, in which he gives several cases of such activities. A man named Harpagus killed a hare and hid a message in its belly. Then he sent the hare with a messenger who pretended to be a hunter. [DaKah96]

To inform his friends that it was time to begin a revolt against the Medes and the Persians, Histaieus shaved the head of one of his trusted slaves, tattooed the message on his head and waited till his hair grew back. After that, he sent him along with the instruction to shave his head and his friends received the message. [DaKah96][Alj98]

Another technique was using the tablets covered by wax. Herodotus also tells about Demeratus, who wanted to report from the Persian court back to his friends in Greece that Xerxes the Great was about to invade them. He hid this by hiding the messages under writing tablets. In that period the writing tables were usually two pieces of wood covered with wax, hinged as a book. One wrote on the wax, the recipient melted the wax and reused the tablet. The technique that Demeratus used was to remove the wax, to write his message on the wood and then to re-cover the wood with wax. The tablets were sent then as apparently blank tablets to Greece. In the beginning, this thing worked, but after a while a woman named Gorgo guessed that maybe the wax hides something, so she removed the wax and became the first woman cryptanalyst. [DaKah96]

Aeneas the Tactician, who wrote on military matters, invented the astrogal. He drilled holes in a cube of material (wood), each hole representing a letter. In order to spell the message, he passed a thread through the holes. The recipient had to disentangle it and note the successive holes through which the thread passed. This astrogal was regarded as a toy when it was intercepted. [DaKah96]

During the Renaissance, Harpagus' hare technique was "improved" by Giovanni Porta, one of the greatest cryptologists of his time, who proposed feeding a message to a dog and then killing the dog. [DaKah96]

Drawings were also used to conceal information. It is a simple matter to hide information by varying the length of a line, by shadings or other elements of the picture. [Alj98]

Another interesting and widespread technique was the use of sympathetic inks. Who have not heard about lemon-based inks during childhood? Using such inks it is possible to write an innocent letter having very different messages written between lines. Those inks are very old and were described by Pliny the Elder as far back as the first century AD Also Ovid in his "Art of Love" suggests using milk to write invisibly. To "decode" the message, the recipient sprinkles soot or carbon black on the paper and on the paper and it will stick to the milk residue. The initial inks were only simply organic fluids which being heated very gentle with a candle would reveal the secret message. [DaKah98] [Alj98]

During the World War 1, the Germans put almost invisible pinpricks above the letters in magazines. They also dotted letters with invisible inks and when heated the plaintext letters could be seen. [DaKah98]

As a result of global progress of the science there were developed chemically sympathetic inks (chemicals that could be combined with other chemicals in order to cause a reaction that would make the result visible). The chroniclers mention such techniques since the classical times. One of them is gallotanic acid made from gall nuts, which will became visible if copper sulfate is painted over it. Another example is much closer to our times. During the World War 2, the Nazi spy George Dasch wrote messages on his handkerchief using a solution of copper sulfate too, which remained invisible until it was exposed to ammonia fumes. During the World Wars 1 and 2, chemicals were developed that reacted with very specific chemicals in order to produce something visible. The messages using such techniques must be 'developed' much as the photographs are developed with a number of chemicals in processing labs. Censors tried to discover these secret inks by taking multiple brushes that were wired together, dipping them in some general reagents and 'striping' a letter suspected of secret inks with them. This thing worked for most of the simpler inks. More sophisticated ones were used and these were almost impossible to intercept. For example, secret ink based on a compound very widely used in laxatives: phenolphthalein. [DaKah96]

Photography allowed great reduction in images, so a page could be made very small. One such example is in the Franco – Prussian war, when Paris was under siege and people wanted to send messages outside. So they took photographs of their letters, reduced them to images about an inch by half an inch on the film, wrapped the film around the legs of the pigeons and freed them to fly out of Paris. Such examples can be found at the postal museum in Paris.

With the continuous improvement of the lenses, photo cameras and films, people were able to reduce the size of a photo down to the size of a printed period. One such example is the *microdot* technology, developed by the Germans during the World War 2, which FBI director J. Edgar Hoover referred to as "the enemy's masterpiece of the espionage". Microdots are photographs the size of a printed period, having the clarity of standard sized typewritten pages. The first microdots were discovered masquerading as a period on a typed envelope carried by a German agent in 1941. The message was not hidden, nor encrypted. It was just so small as not to draw attention to itself (at least just for a while). Being so small, the microdots permitted the transmission of a large amount of data, including drawings and photographs. [NeJoh98]

With many methods being discovered and intercepted, the authorities took extreme measures such as banning flower deliveries, which contained delivery dates, crossword puzzles. The censors even went so far as rewording letters and replacing stamps on envelopes. [NeJoh98]

Spread spectrum communication is the process of spreading the bandwidth of a narrowband signal across a wide band of frequencies. Successive brief portions of the message are sent over a pre-determined sequence of frequencies. The recipient must know the correct succession of the frequencies. The advantage is that the portion sent over a certain frequency is so small that a monitor will usually not perceive it.

There are also other forms of hidden communications, like the semagrams and the open codes. [DaKah96]

A semagram is a secret communication that is not in written form. For example, during the World War 2 censors intercepted a shipment of watches. They changed the position of the hands fearing that the position of the hands of the watches could be spelling out a secret message.

Open codes use illusion or code words. In World War 1 the German spies used fake orders for cigars to represent various types of British war ships – cruisers and destroyers. For example 5,000 cigars needed in Portsmouth meant that there were five cruisers in Portsmouth. A woman named Valerie Dickinson used dolls to represent the Americans vessels in the messages she sent to the Japanese. Small dolls would represent destroyers and large dolls might stand for aircraft carriers or battleships.

Null ciphers were also used. Using such technique the real message is "camouflaged" in an innocent sounding message. The messages are very hard to construct and sound always very strange. Due to this fact, mail filters detected the suspect communications. However "innocent" messages were allowed to pass through. The strangeness can be reduced if the constructor has enough space and time. A famous case of a null cipher is the book "Hypnerotomachia Poliphili" of 1499, when the secret message was spelt out as the first letter of the each chapter; this is an example of a message constructed using plenty of room to submerge the covertext. The message was "Father Colona Passionately loves Polia". It was not good for a Catholic priest to be admitting this kind of thing. But after a few years when the message was discovered, on the book having no author's name on the title page, Colona was discovered. Thomas Usk – an English writer – used the same technique to conceal the authorship. But selecting letters of successive words to form a message always sounds funny and raises suspicion.

An example of a text containing a null cipher is: [NeJoh98]

"News Eight Weather: tonight increasing snow. Unexpected precipitation smothers eastern towns. Be extremely cautious and use snowtires especially heading east. The highways are knowingly slippery. Highway evacuation is suspected. Police report emergencies in downtown ending near Tuesday"

By taking the first letter of each word, the following message can be discovered:

"Newt is upset because he thinks he is president".

Another technique is to use the second letter in each word:

"Apparently neutral's protest is thoroughly discounted and ignored. Isman hard it. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetable oils".

The message encoded is:

"Pershing sails from NY June 1".

This example is a null cipher message sent by a German spy in World War 2.

Another interesting technique was the Cardan Grille. In a sheet of cardboard there are cut word-length holes. The encoder places the grille on a sheet of paper and writes the secret message in the holes. He then removes the grill, fills the empty spaces between the words and sends the letter. The recipient places his grille (a copy of the grille used by the encoder) and reads the secret message. The problem is that constructing cover messages is not always an easy task. The grille messages sounds also funny, so they are detected like the null ciphers.

An example of applying this scheme is the following test: [NeJoh98]

"We **explore** new steganographic and cryptographic algorithms and techniques throughout **the world** to produce **wide** variety and security in the electronic **web** called the Internet".

Using a grille having cut the bold words, the result is:

"explore the world wide web"

Another technique derived from the grille scheme is shifting imperceptibly some words. Using the unmodified version together with the shifted one and placing one above the other, we will obtain something like this:

> “`We` **`explore`** `new steganographic and cryptographic algorithms and techniques throughout` **`the world`** `to produce` **`wide`** `variety and security in the electronic` **`web`** `called the Internet`”.

The modified copy, obtained by expanding the space before explore, the, world, wide, web by one point and condensing the space after those words by one point replaces the grill. Independently, the two copies appear harmless, but combining them produces the message “*explore the world wide web*”.

During the World War 2, US Marines in the Pacific frequently called on Navajo Indian “codetalkers” to “encrypt” secret messages transmitted by radio using their notoriously difficult and obscure native tongue. It was thought that only 28 non-Navajos could speak the language and none of those were German of Japanese. To make it more difficult for potential eavesdroppers, the codetalkers spoke a slangy, cryptic lingo that even Navajo speakers unfamiliar with the jargon could not understand. The Germans and the Japanese did not stand a chance. [Alj98]

Nowadays the steganography is not forgotten; there are many real life applications of it. Apparently, during the 1980’s, the British Prime Minister Margaret Thatcher became so irritated at press leaks of cabinet documents that she had ordered that the word processors encode their identity using word spacing, so that disloyal ministers could be traced.

The US and the USSR wanted to place sensors in each other’s nuclear facilities that would transmit certain information (such as the number of the missiles) but not reveal other kind of information (such as their location).

Steganographic software is very new but its potential for hiding data is awesome. For example, the images are represented in computers as an array of numbers. One of the modifications we can make is to replace the least significant bit of the original image with the secret bits and the image will not change – most graphics standards specify more gradations of color than the human eye can notify. You can store a 64 kilobytes message in a 1024 x 1024 grayscale picture this way.

We have seen that concerns about communicating via hidden channels exist since a very long time. We have seen that the methods used for hiding information evolved from very simple techniques like writing underneath the wax of writing tablets to imperceptible modifications that can distinguished only by certain persons, who have the clue. In the next chapters we will present a general overview of the data embedding process and then some specific aspects concerning the implementation of the method for specific file formats.

Chapter 2. Information Hiding in Volatile Data

In this chapter we will see how data can be hidden in the control data and in timing properties of a packet that moves inside a network. The objective is to demonstrate that the covert channels exist through the network model architecture. The users of a computer network should understand that their network systems might be effectively subverted using a wide variety of methods and in a large number of locations.

2.1. An Overview of Data Hiding in the OSI Network Model

The OSI model is a standard network model that is used for almost all the networks to be compared to. It does not exist *per se* in functional systems. The most useful purpose of this model is to break the complex computer networks into parts that can be easily understood.

We will use the Prisoners' Problem in order to discuss possible hidden data communications that can arise inside the OSI model. We have the two prisoners, Alice and Bob, who are allowed to communicate overtly, which is permitted, and covertly, which is absolute banned. They communicate using a network computer. The manager of the network is Willie, the warden, who can monitor the message traffic.

Inside a computer network some techniques work, but some do not. Multiple techniques used inside a network will assure that if one is discovered the others may remain secure. There are some tests for finding the system resources that can be used for covert channels, hiding data. The quality of the covert channel can be expressed in terms of indistinguishability, bandwidth.

The designers of the computer networks must realize that covert channels and data hiding locations exist and are *inherent* components of the information system. Security can deal with known vulnerabilities, but also with unknown functionality. Unknown functionality exists for a number of reasons: design errors, upgrades, product deadlines, after-market modifications, etc. all contribute to creating it. The effectiveness of security is limited by cost-performance tradeoffs. [ThHan96]

2.2. Short Description of the OSI Model

The Open Systems Interconnection Reference Model deals with connecting open systems, that is systems that are open for communication with other systems. [AnTan96]

The OSI model itself is not a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do.

A simplified overview of the OSI reference model is shown in Figure 4.

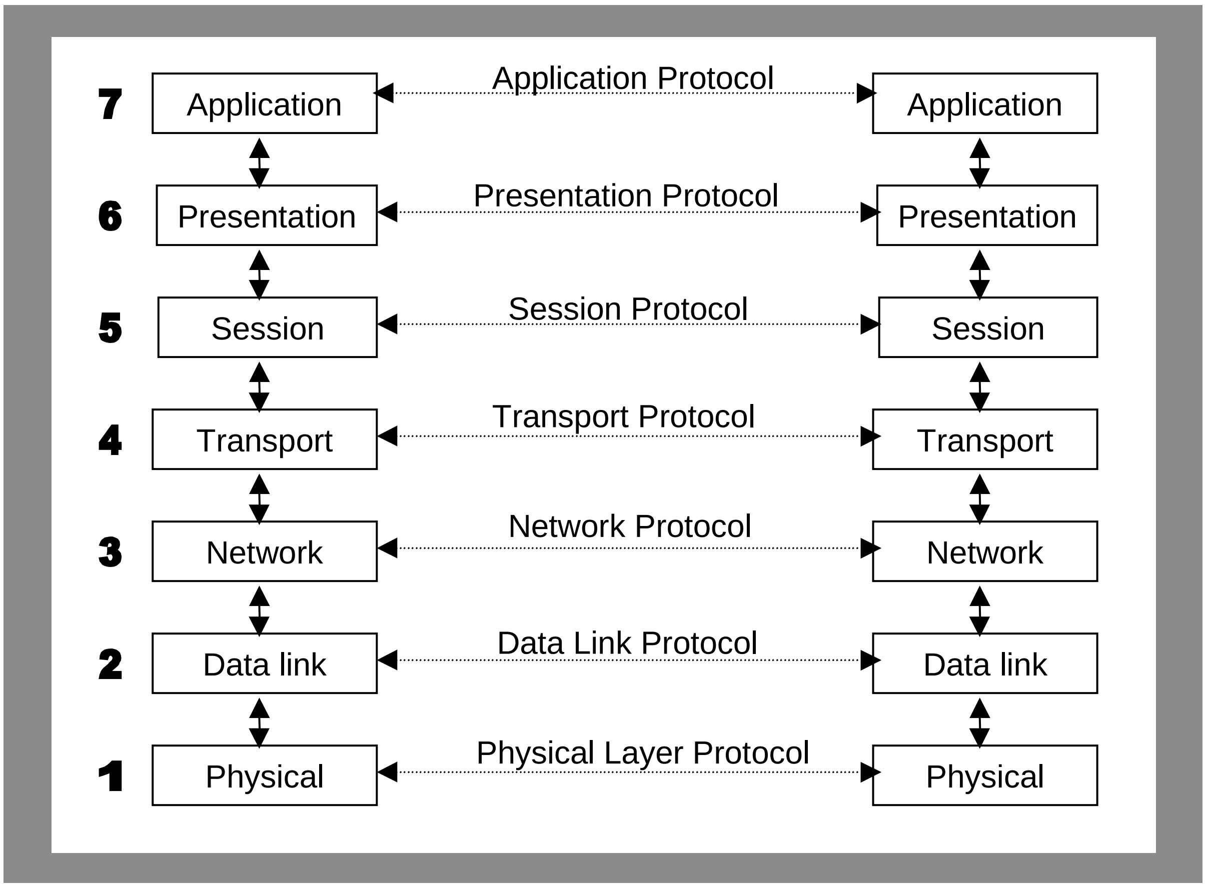


Figure 4. The simplified reference OSI model.

The *physical layer* is concerned with transmitting raw bits over a communication channel. The design issues have to do with making sure that one side sends a 1 bit it is received by the other side as a 1 bit, not as a 0 bit.

The main task of the *data link layer* is to take a raw transmission facility and to transform it into a line that appears free of undetected transmission errors to the network layer. This task is accomplished using *data frames* sent by the sender and *acknowledgement frames* sent back by the receiver.

The *network layer* is concerned with controlling the operation of the subnet. The main design issue is to determine how packets are routed from source to destination, using static or dynamic tables.

The main function of the *transport layer* is to accept data from the session layer, to split it up into smaller units, pass those to the network layer and assure that the pieces all arrive correctly at the other end.

The *session layer* allows users on different machines to establish *sessions* between them. Such a session permits ordinary data transport, like the transport layer does, but also it provides enhanced services useful in some applications.

The *presentation layer* performs certain functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problems. We can say that this layer is concerned overcoming the architectural differences in data representation. For example, an integer value should still be the same value when it arrives at the destination, although the byte order might be different (for example little endian and big endian).

The last layer, the *application layer*, contains a variety of protocols that are commonly needed, such as e-mail, file transmission, remote login, information services, etc.

2.3. Data Hiding in OSI Layers

In the OSI layers described above there are some places where secret information may be hidden. We will present some possible scenarios for introducing covert channels within some layers.

The Physical Layer

At the *physical layer* the transmission of data through the network's communication channels is done using special hardware that can be different depending on the network. The communication is done with the help of control signals and timing.

The baud rate of the serial communication port can be adjusted over a wide range and this adjustment generally defines the channel capacity. There is a handshaking mechanism that controls data flow. Our two prisoners may introduce some latitude of control over the throughput by manipulating the tension of the lines. A "1" is represented by a 5 volts tension. By modifying this tension to 5.5 volts the receiver sees a "1" hidden and if the tension is 5 volts, there is a "0" hidden. A very important aspect is that using the physical layer as a covert channel is generally only possible when no repeater, bridge, routes, etc. is in between the sender and the receiver.

Also the two prisoners can establish covert channels by manipulating the *Clear to Send/Ready to Send* control lines (as in the RS-232 protocol) as shown in Figure 5. [ThHan96]

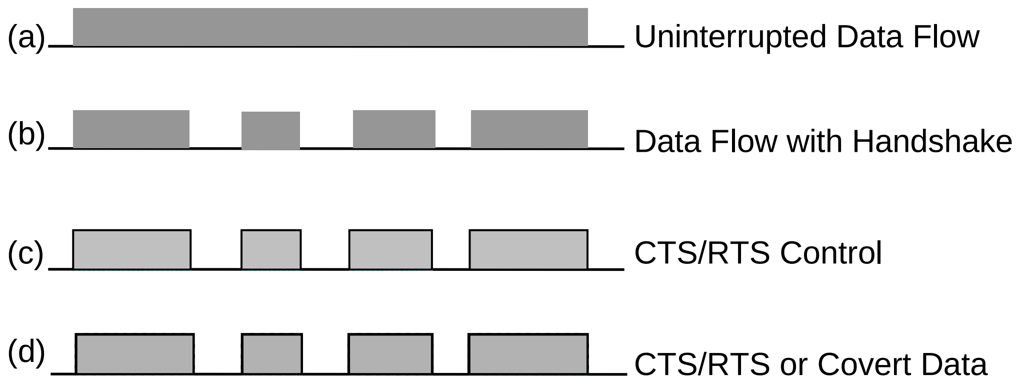


Figure 5. Waveforms at the data layer.

By analyzing the communication channel Walter cannot know if the waveforms he sees are "normal" signals or are signals used by the two prisoners for communications using for example Morse Code.

The Data Link Layer

At this layer, the two prisoners can modify the collision detection system (for example from Ethernet) by setting the delay of packet's retransmission to either zero or maximum (any packets generated by a third user can be jammed by one of the prisoners). The second prisoner is able to notify the order of packets and so we have a bit-per-packet hidden channel. The warden must determine if there are deliberately jams. Note that this technique does not require a direct connection between the two prisoners. [ThHan96]

The Network Layer

The *network layer* adds supplementary information such as headers used for routing information and possibly error control. This layer may also fragment packets at the sender and reassemble them at the receiver.

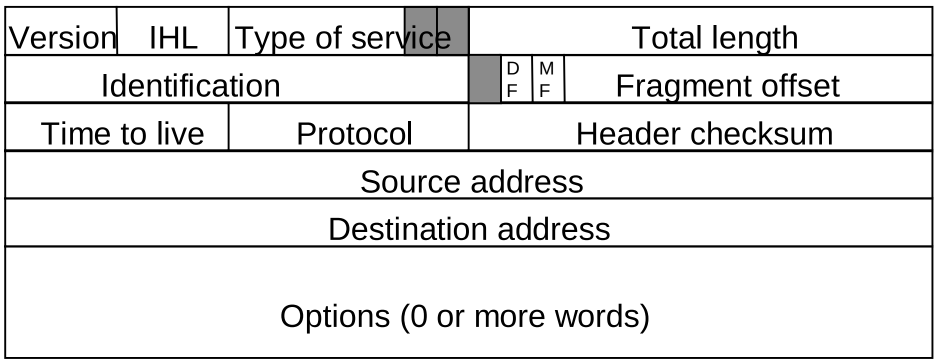


Figure 6. The IPv4 header.

In Figure 6 we have shown how the IP header (version 4) is organized and we have marked three unused bits. One is before the DF and MF bits and another unused portion of this header is inside the *Type of service* field where we have two unused bits (the least significant bits). All those bits can be used to store information, that is, to create a covert channel. This technique is similar to hiding information on systems that do not occupy all "real" disk space, for example on floppy-disks one can store information beginning from the end of the media and writing towards the valid data. The IP version 6 packet header does not have such unused bits, being more compact with a fixed length of 40 bytes.

Another method is to use the *Internet Message Control* protocol (ICMP), as suggested in [ThHan96]. The source-quench command adjusts the sending device data rate when the destination or the intermediate node cannot keep up with the rate. The flow control in this situation is very similar to the CTS/RTS implementation in hardware.

A similar scenario is that the receiver requests the retransmission of the every ten packet. If he does so, the sender "sees" a "1" and if the tenth packet is not requested then there is a "0". By combining this with an error detection/correction mechanism we improve the performance.

The packets may be fragmented at this layer. By choosing the offset of the fragments, the prisoners can communicate by choosing for example the last bit of the *Fragment Offset* field as being the hidden bit (or the last two bits). The sender splits the packets and he choose the length of the packets accordingly to the secret bits he wants to embed.

The Transport Layer

The Internet implementation for the *transport layer* is the *Transmission Control Protocol* (TCP). Every TCP segment begins with a fixed-format 20-byte header. The 13th and the 14th bytes are shown in Figure 7.

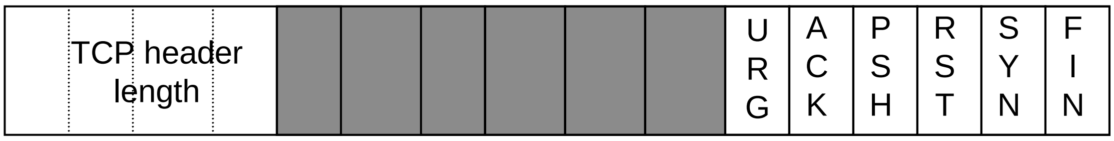


Figure 7. The 13th and 14th bytes of the TCP header

There is a 6-bit field that is not used. Techniques that are similar to those discussed for the network layer can be used for data hiding.

The Session Layer

The *session layer* allows users to access a network, establishes connections between processes on different hosts, thus the name "session". This functionality is achieved by using software that can "mount" remote discs on a local machine. The two prisoners can communicate using this. Supposing we have two files on the disk of Alice, Bob can read one of them. If he reads the first file then Alice records a zero and if he reads the second file she records a one. Willie can see the traffic, but the fact that Bob reads one or the other file is not relevant for him. There was a demonstration of this scheme, obtaining a 300 BPS rate. [ThHan96]

In the next chapter we will present techniques that embed information in the user data.

Chapter 3. Information Hiding in User Data

User data is what the sender wants to transmit to the receiver; this information can be split during the transmission within the network, but at the receiver it is reassembled. Unlike the control data and timing properties, this data is not volatile: it is stored on the user's harddisk or in memory, etc. until he decides to erase it. So it is possible to prove later by a third person that two users have communicated secretly if the data still exists.

3.1. A Framework for Steganographic Algorithms

The steganographic algorithms are based on replacing a noise component of a digital object with a pseudo-random secret message. According to the terminology proposed by Pfitzmann to the First International Workshop on Information Hiding [BiPfi96], we call the noisy message *cover* and the bits carrying the noise *cover bits*. The bits embedded as pseudo-random noise are *secret bits*. The cover bits substituted with the secret bits are called *hiding bits*.

The cover messages can be divided in two classes. [TuAur96] The cover may be a *stream cover* or a *random access cover*. The former is a contiguous data stream, like a telephone connection and the latter may be a file, for example a wave file. In a stream cover it may not be possible to tell in advance where the cover message begins, when it ends or how long it will be. One may not know in advance what the next bits of the cover will be, because those bits are selected using a keystream generator, that gives the distance between two consecutive bits in the stream. This is the *random interval method*. The receiver has to synchronize his keystream generator with the message in order to recover the secret data. The random access covers do not have the same disadvantages as the stream covers, because the sender knows in advance the file size and its contents. The information can be inserted in any particular order, for example the header may be inserted after the secret message. The drawback of random access covers is that their length is shorter than a data stream.

As a general principle, in order for a slight increase in the cover data's randomness should be noticeable, it must be sufficiently noisy. Typically, the cover bits are the least significant bits (LSB) of the digital object. These bits are the result of some series of measurements. For example in sounds the digital information can be viewed as a huge array of measurement results, each component having a random error component.

The least significant bits may have some statistical properties. Changing some of them could result in the loss of those properties. Thus the message should be embedded mimicking the characteristics of the cover bits. One possibility is to generate a large number of cover messages in the same way and to choose the one having the secret embedded in it. This method is called *selection method*. For example, a photo can be scanned until the digital representation has incidentally the desired properties. The file has the statistical properties that any other digitized image has. The drawback of this method is that it can hide only a small amount of data with lots of work. Another possibility is to build a mimic function that has the characteristics of the cover bits noise. This method is called *constructive method*.

Generally, both the sender and the receiver share a secret key and use it with a keystream generator. The keystream is then used for selecting the positions where the secret bits will be embedded. Not every pixel may be suitable for encoding ciphertext. Changing the values of some pixels may result in a degradation of the quality of the original object. For example in digital imagery not any pixel is suitable for being changed. There are algorithms that can decide if a pixel may be changed by checking the variance in luminosity of the surrounding pixels that may be neither very low nor very high.

Although the LSB embedding methods hide data in such a way that the humans do not perceive it such schemes can be easily destroyed by an opponent, using for example lossy compression algorithms or a filtering process. Any process that performs, directly or indirectly, a low pass filtering can easily eliminate the data placed on the high frequency spectrum. One possible countermeasure is to use error correction codes or to hide data in more than one location.

Another interesting phenomenon that can be used for embedding data is the *masking* phenomenon [RoAnd98]: one sound interferes with other sound. *Frequency masking* occurs when two tones that are closed in frequency are played at the same time. The louder will mask the quieter. *Temporal masking* occurs when a low-level signal is played immediately before or after a stronger one. After we hear a loud sound, it takes a little while before we can hear a quiet one. Just like the previous schemes, those facts are also exploited by the MPEG audio compression techniques. Of course the design of compression schemes is limited in most cases by economic factors: the amount of computation we are prepared to do is not infinite. If we are prepared to do a little bit more work than the "ordinary" user then we will be able to exploit a number of low-bandwidth stego channels.

The conclusion of the above discussion is that it is better to insert the data into the perceptual significant regions of the digital object. The problem is now finding a scheme for embedding data without the alterations of the object become noticeable. We can make an analogy between such schemes and spread spectrum communications. In the latter one transmits a narrow band signal over a larger bandwidth in a way that the signal energy present in any single frequency is imperceptible. The secret bits are also spread over many frequencies, so that the energy in every single frequency is very small and undetectable. The process of destroying the data inserted would require noise to be added to all frequencies, resulting in altering the original data. Spreading the secret bits throughout the spectrum of an image ensures a measure of security against unintentional or intentional attacks. The spatial location of the secret bits is not obvious, the frequencies for hiding information may be chosen in such a way that any attack will have as a result the degradation of the original cover object. A mark embedded will not be noticeable because the energy in any single frequency is sufficiently small. Moreover, using the masking effect presented above it is possible to increase the amount of the hidden data in some particular frequencies.

Concerning the extraction of the data inserted, the schemes can be classified as follows: *cover escrow schemes* and *blind schemes*. [LiMar98] Schemes where the original cover signal is needed to reveal the hidden information are known as cover escrow. Blind, or oblivious, schemes allow direct extraction of the embedded data from the modified signal without knowledge of the original cover. The former schemes are not practical, blind strategies being predominant among steganography of the present day.

A general scheme for embedding data is depicted in Figure 8. A message is embedded in a file by the stegosystem encoder, which has as inputs the original cover, the secret message and a key. The resulting stego object is then transmitted over a communication channel to the recipient where the stegosystem decoder using the same key processes it and the message can be read.

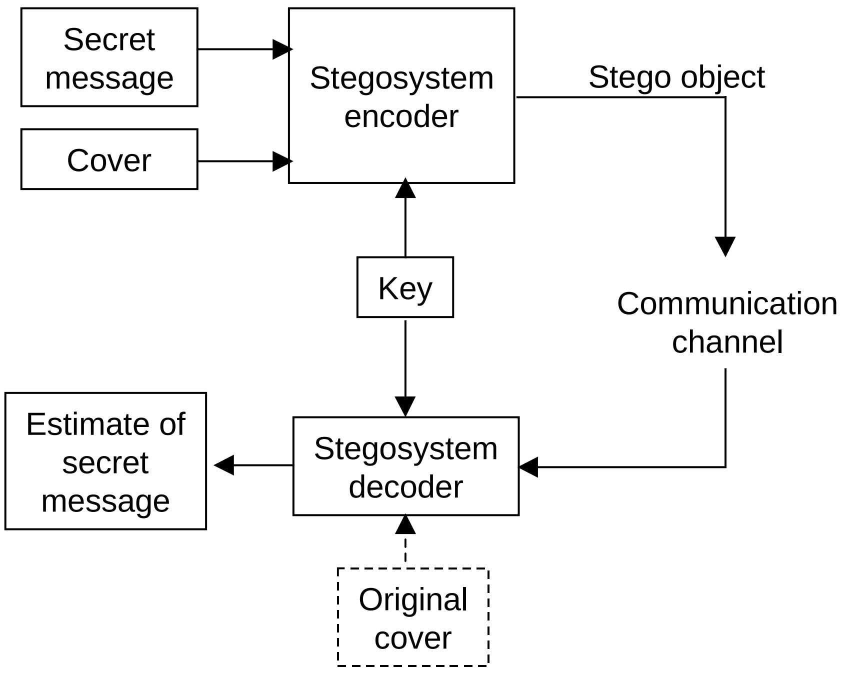


Figure 8. A General Steganographic Model.

The steganographic process can be represented using formulas. The stego object is given by:

$$I' = f(I, m, k)$$

where: *I'* is the stego object
I is the original object
m is the message
k is the key that the two parties share.

The stego object may be subject to many distortions, which can be represented as a noise process *n*:

$$I'' = I' + n(I')$$

At the decoder we wish to extract the signal *m*, so we can consider the unwanted signal to be *I*. The embedded signal should resist common signal distortions as those depicted in Figure 9.

Two kinds of compression exist: *lossy* and *lossless*.[SuJaj98] [MaSan95] Both methods save storage space but have different results. Lossless compression permits exact reconstruction of the original message; therefore it is preferred when the original information must remain intact. Such compression schemes are the images saved as GIF (Graphic Interchange Format) or BMP (Microsoft Windows bitmap file). Lossy compression, on the other hand, does not maintain the original's integrity. Such compression scheme is an image saved as JPEG (Joint

Photographic Experts Group). The JPEG formats provide close approximations to high-quality digital photos but not an exact duplicate.

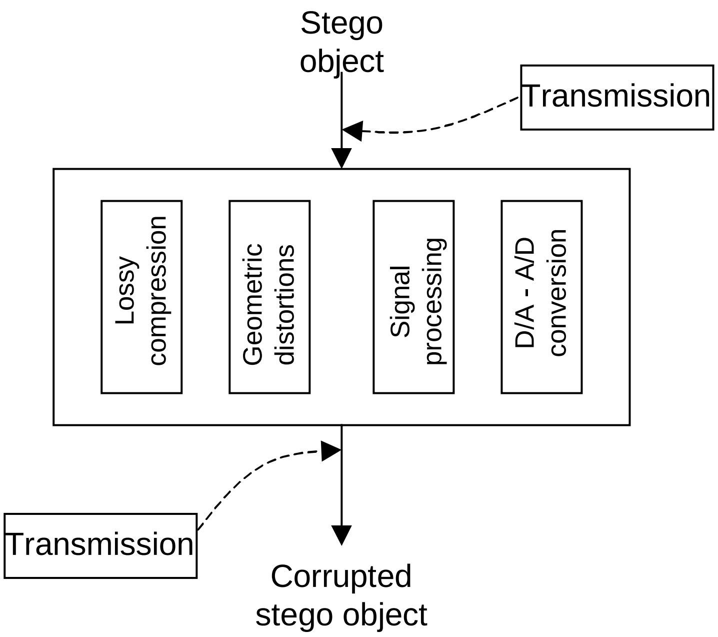


Figure 9. Common signal distortions over the transmission channel.

Geometric distortions are specific to images and video and include operations such: rotation, translation, scaling. Cropping is another type of geometric distortion that leads to irretrievable loss of data.

The signal can be processed in many ways, including resampling, analog-to-digital and digital-to-analog conversions, requantization, recompression, printing, rescanning, copying, filtering and so on. All those processes introduce additional degradation into the object that a steganographic scheme must withstand. Of course there is a possibility that a combination of those above distortions should be applied to the watermarked signal.

In the next part we will present some techniques for embedding data into various number of cover types: document images, sound files, image files and binary files.

3.2. Information Hiding in Document Images

Electronic distribution of publications is today available through CD-ROMs, on-line databases, computer network based search engines, electronic libraries, etc. There is an electronic library, the *Right Pages*, developed by Bell Laboratories and installed at the University of California in San Francisco.[JaBra94] Electronic publishing is preferred because of the low cost of text processors (computers) and the high quality of displays and printers.

In order for electronic publishing to become accepted, the publishers must be assured that their benefits will not be diminished or lost due to theft of copyrighted materials. While photocopy infringements of copyright have always concerned publishers, the need for document security is much greater for electronic document distribution. While originals and photocopies of a paper document can look slightly different, copies of electronic material are identical.

The techniques used for data hiding in document images embed *marks* in every copy of a document. Those marks can be the same for all the copies or may differ.

Altering the text formatting or the characteristics of the characters can achieve document marking (watermarking). The main purpose is to make alterations to the text in such a way that those alterations are not visible but can be decodable. The alterations may be applied to an image representation of a document or to a document format file. The latter is a computer file describing the content of the document and its layout; from this specification is generated the image that the reader sees. The image representation describes each page of the document as an array of pixels. We can consider that the image of a page is represented by a function:

$$f(x,y)=0 \quad or \quad 1 \quad , \qquad x \in [0,W] \quad , \qquad y \in [0,L]$$

where W and L depend on the scanning resolution, representing the width and length of the page. The image of a text line can be represented using a similar formula:

$$f(x,y)=0 \quad or \quad 1 \quad , \qquad x \in [0,W] \quad , \qquad y \in [t,b] \qquad (1)$$

where t and b are the top and the bottom of the text line. The codeword assigned to a particular document specifies what lines to be moved in the recipient's document. The decoding performance is improved if we are using a *differential encoding technique*. With this coding scheme we keep the first, the third, the fifth, etc. lines unmodified, and the second, the fourth, the sixth, etc. lines are modified. This encoding is resistant to image defects that can appear, accidentally or not, during the transmission process. There are three major techniques for hiding information in documents: *line-shift coding, word-shift coding* and *feature coding*. We will present them in detail.

A good document marking process has to respect some characteristics:

- A text line is marked only if it and its two neighbors are sufficiently long. The adjacent lines are the control lines, being unmodified.
- One line is marked both using line shifting and word shifting. In line shifting the line is shifted slightly up or down from its normal position and in word shifting we have three blocks of words, the middle block being shifted left or right. The unique identifier determines the direction of the shifting.

The process of detection of the embedded mark requires:

- Scanning in the recovered copy if its digital representation is not available.
- Compiling the horizontal and the vertical profile of the copy.
- Compensating the most common distortions that a document can be exposed to (if it is possible).
- Detecting the lines and/or word shifting
- Recomposing the mark

Figures 10 – 14 are derived from [JaBra94] [JaBra95].

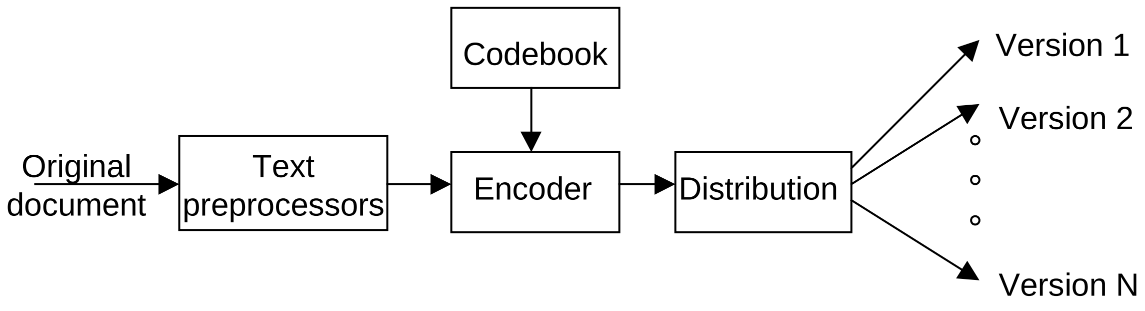


Figure 10. The architecture of an encoder.

The data hiding techniques require an *encoder* and a *decoder*. The encoder receives as input the original file and produces the marked file as output, as shown in Figure 10. The original document is preprocessed and the encoder receives a virtual page, in which it will do the modifications, according to the code choose from the codebook. The output of the encoder is the modified file that is then distributed (for example printed or saved on a floppy). The decoder has as input a file and must produce as output the mark embedded in it. Figure 11 describes this process.

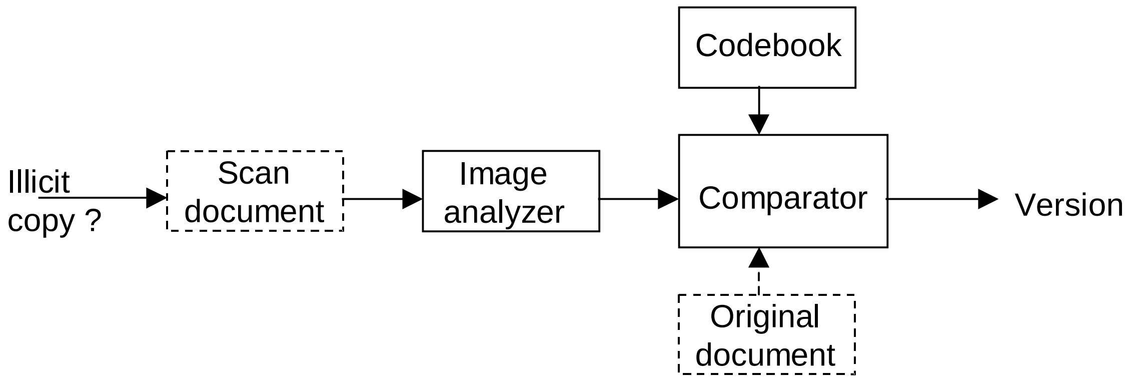


Figure 11. The architecture of a decoder.

First the digital form of the document is needed. Then the image is analyzed and compared to see if there is a mark in the codebook that fits. The result is the mark found inside the document. A presentation of those techniques is following.

3.2.1. The Line-Shift Coding

This technique alters a document by vertically shifting the position of the locations of text lines and may be applied to both the page image and the file format. The codeword preassigned for a certain document specifies the text lines that will be moved in that document. We may have a “0” for a line shifted up and a “1” for a line shifted down. But also we may have a “-1” for a line shifted up, a “0” for an unmoved line and a “+1” for a line shifted down. This technique uses the differential encoding technique for achieving performance and robustness. The length of each codeword that can be hidden is reduced, comparing to the technique that shifted every line, but the number can still be large. For example, having a page with 40 lines, that is $2^{20} = 1{,}048{,}576$ distinct codewords per page.

The encoder has to shift the lines up or down, according to the mark wanting to embed into the file. The decoder measures the distance between each pair of two neighboring lines. This can be done using two different techniques: either the decoder measure the distance between the *baselines* of adjacent lines or the decoder measures the distance between the *centroids* of two adjacent lines. A baseline is a logical line on which the characters of a line sit; a centroid is the center of mass of a certain text line.

Supposing that the text lines *i-1* and *i+1* are not shifted and the line *i* is shifted either up or down. In an unaltered document file the distance between the baselines of two adjacent lines is constant. Now let s_{i-1} and s_i be the distances between baselines *i-1* and *i* and between baselines *i* and *i+1,* respectively. The algorithm uses a simple detection rule as follows:

```
if $s_{i-1} > s_i$ then     line $i$ is shifted down
if $s_{i-1} < s_i$ then     line $i$ is shifted up
otherwise                   uncertain.
```

Unlike baseline spacing, centroid spacing may not be necessarily uniformly spaced. In methods that measure the distance between centroids the decision is based on the difference between centroid spacing in the original document and in the altered document. To calculate the position of centroids, we can use the following formula:

$$c_i = \frac{\sum_{j=t_i}^{b_i} [j \cdot n(j)]}{\sum_{j=t_i}^{b_i} n(j)} \qquad (2)$$

where *i=1..N*, is the current line,
N is the number of lines on the page,
t_i and b_i are top and bottom limits of the line *i*,
n is a function that counts how many pixels are ON (*f(k,j)=1, k=0…W*, see (1))

The next step is to calculate the distance between the centroids of the lines *i-1* and *i*, and *i* and *i-1*, let it be s_{i-1} and s_i. Similarly, the decision may be:

```
if $s_{i-1} - t_{i-1} > s_i - t_i$ then     line $i$ is shifted down
otherwise                                   line $i$ is shifted up.
```

Figure 12 shows a fragment of a document encoded using line shifting coding. In the second part, the second line was moved with 0,1 mm.

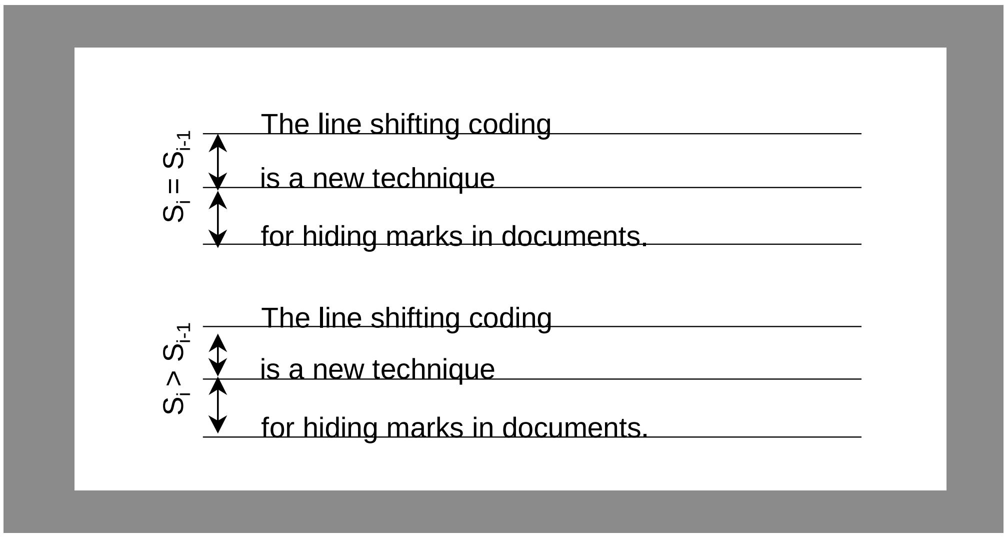


Figure 12. An example of line shift coding.

3.2.2. The Word Shift Coding

This scheme alters the document by horizontally shifting the locations of words within text lines to embed the unique mark. The space between adjacent words must be different in order to apply this technique. Variable word spacing is commonly used to distribute white space when justifying a document (as is this paper). Because of the variable spacing, the decoder needs the original document or a specification about word spacing in the original document.

The encoder first determines if a line has sufficient number of words to encode; short lines are not encoded. On each encodable text line found is applied the differential encoding technique for this scheme. The second, fourth, sixth, etc. word from the left margin is displaced. The first and the last word on each line are unshifted to maintain the column justification. After the process of shifting words is finished, the document is distributed.

The decoder needs information about the original document. This is not a drawback, knowing the fact that in general the authors are tracing their documents and they own a copy of the original document. The information needed is the position of the beginning of each word or the position of centroids for each word. The centroids' positions are calculated using a similar formula, as in (2). Supposing that the i^{th} word was shifted and the original centroids' positions are c_{i-1}, c_i and c_{i-1} and the modified centroids' positions are c'_{i-1}, c'_i and c'_{i+1} then the algorithm calculates:

$$d_l = c_i - c_{i-1} \qquad d'_l = c'_i - c'_{i-1}$$
$$d_r = c_{i+1} - c_i \qquad d'_r = c'_{i+1} - c'_i$$

and finally will decide if the word was shifted left or right accordingly to the next statement:

```
if $d'_l - d_l < d'_r - d_r$ then    the word i was shifted left
if $d'_l - d_l > d'_r - d_r$ then    the word i was shifted right.
```

In Figure 13 we outlined an example of this technique.

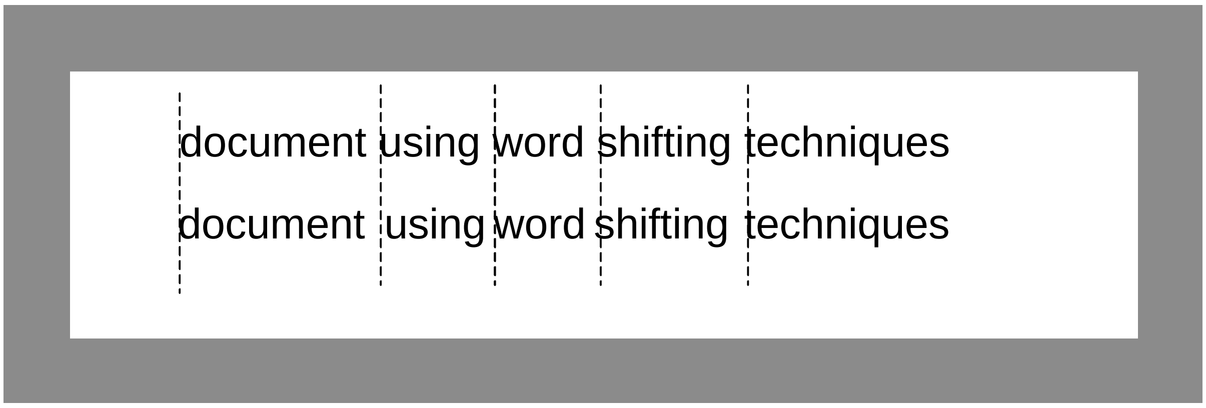

document using word shifting techniques
document using word shifting techniques

Figure 13. An example of word shifting encoding (exaggerated).

A slightly different method is described in [WaBen96]. The data is encoded using extra spaces added for text justification. It uses the Manchester encoding in order to place hidden bits into the file. Using the Manchester encoding, a “01” sequence is decoded as a “1” and a “10” sequence is a “0”. An example of such a technique is in Figure 14.

```
Modern__computer_networks__make_it_possible __to__distribute_documents_quickly_and_  | __⇒_ 10⇒1
economically.__This_is_because_of__the_decreasing_cost_of_the_equipment__needed_to_ | _⇒__ 01⇒0
copy,_to_print,_to_process_the_information.                                         |
```

Figure 14. Data hiding through justification.

3.2.3. Feature Coding

According to this scheme, the image is examined for chosen text features and those features are altered, depending on the mark inserted. Such features may be the vertical lines of the letters *b*, *d*, *h*, *k*, etc. The length of those lines may be modified in a way that is imperceptible to the ordinary readers. The character heights within a given font may also be changed.

There are also techniques that change the words themselves substituting them with synonyms. Usually there are two pairs of synonyms and using one or the other synonym is equivalent with embedding a “0” or a “1”. The two parties involved must share the synonymous pairs.

The difference between those two techniques is that the first one can be used for embedding copyright information, but the second one only hides information, being adequate in the prisoners’ problem. In the former all documents will have the same content, but some characters will be modified, in the latter two documents having different marks embedded will be different.

We have presented some techniques for watermarking a document in order to prevent illegal copying and to allow the tracing of document files. In the next subchapter we will present techniques for hiding data in sound files.

3.3. Information Hiding in Sound Files

Like the document images, the sound files may be modified in such a way that they contain hidden information, like copyright information; those modifications must be done in such a way that it should be impossible for a pirate to remove it, at least not without destroying the original signal.

The methods that embeds data in sound files use the properties of the Human Auditory System (HAS). The HAS perceives the additive random noise and also the perturbations in a sound file can also be detected. But there are some “holes” we can exploit. While the HAS have a large dynamic range, it has a fairly small differential range. As a result, loud sounds tend to mask out quiet sounds. And there are also some distortions that are so common that the HAS ignores them.

The digital sound is obtained from the analog sound by converting it to digital domain. This process implies two subprocesses: *sampling* and *quantization*. Sampling is the process in which the analogue values are only captured at regular time intervals. Quantization converts each input value into one of a discrete value. Popular sampling rates for audio include 8 kHz, 9.6 kHz, 10 kHz, 12 kHz, 16 kHz, 22.05 kHz and 44.1 kHz. The most popular file formats for sounds are the Windows Audio-Visual (WAV) and the Audio Interchange File Format (AIFF). There are also compression algorithms such as the International Standards Organization Motion Pictures Expert Group-Audio (ISO MPEG-AUDIO).

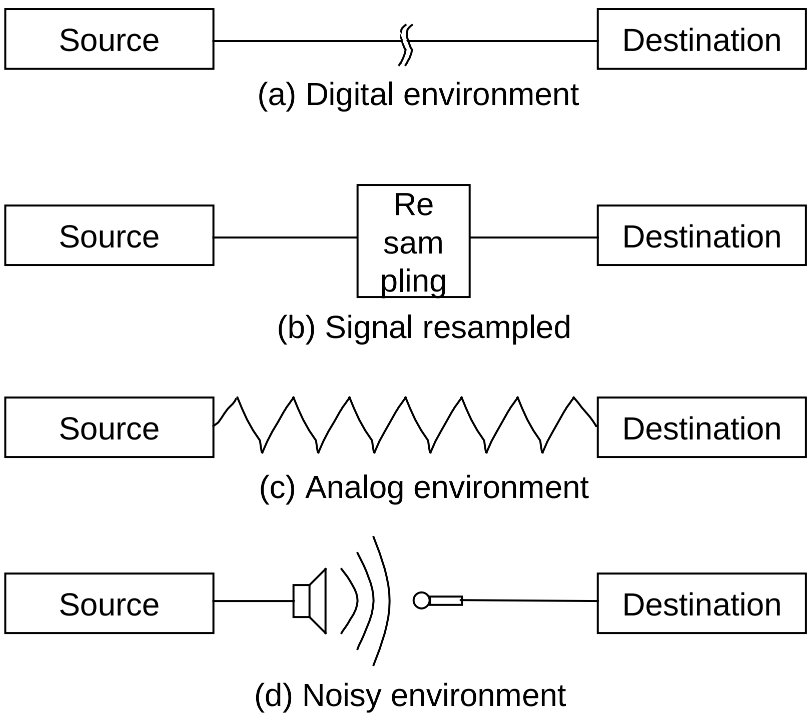


Figure 15. Transmission environments.

Having the sound in the digital format, it may be transmitted over large environment types. In Figure 15 we outlined some possible scenarios. The first signal is transmitted through the environment in such a way that is unmodified. As a result, the phase and the amplitude are unchanged. In the second case, the signal is resampled to a higher or lower sampling rate. The amplitude and the phase are left unchanged, but the temporal characteristics are changed. The third case is to convert the signal and transmit it in the analog form. In this case, even if the analog line is considered clear, the amplitude, the phase and the sampling rate are changed. The last case is when the environment is not clear, the signal being subjected to nonlinear transformations, resulting in phase changes, amplitude changes, echoes, etc.

A common method for embedding data in sound files is to add it as noise. This scheme introduces data in the LSB of the covers. The degradation of the original stream must be minimal, the HAS must not perceive those modifications. Better methods hide data in the significant regions of the signal. In this case, the modifications should not be perceivable and the advantage is that lossy compression algorithms will not remove the data.

We will present further on a method based on embedding data in LSB and two methods for hiding information in significant parts of an audio signal.

3.3.1. A Method for Embedding Data in LSB

Having a stream of bits, this method sets the LSB of the cover accordingly to the values of a second file, that is the file wanting to be transmitted, as described in [ElFra96]. The distance between two successive hidden bits is the number of samples between them and is controlled by a random value. Those distances vary between 0 and a maximum and are a secret between the sender and the receiver. They correspond to a secret key in a symmetric cryptosystem, therefore the Kerchoff's principle is valid. Without knowing the correct succession of those distances between hidden bits, any attack has very little chance to succeed. The distances may be generated all separately or starting from an original value and using a pseudo-random generator having this value as input. The latter method is more elegant, the initial value may be seen as a password shared by the two parties and also has the advantage that only a value must be transmitted between them, not all.

In Figure 16 we describe this scheme. The sender modifies the original stream using the secret key. With the continuous arrows we outlined the position in which the hidden bits will be inserted. The new stream is obtained from the original stream, but is modified according to the values of the stream to be hidden (see the dotted interrupted arrows). For example, if the distance is 2, we let two bytes unchanged and insert the secret bit as the LSB of the third byte. If the distance is 0, then we modify the first byte changing its LSB and so on. This is done on the sender's part. The recipient recovers the hidden bits using the shared key, that gives the positions of embedded bits (see continuous arrows), and extracts them (see the interrupted arrows).

The values of the distances between the successive hidden bits must be between two extreme values. The minimal value is usually 0, but the maximal value must be correctly chosen. If the length of the cover bits is long enough, then there is no problem. But if the length is not long or we have a stream cover (see chapter 2.4), then the sender must take some measures to be sure that all the bits to be hidden will enter in the cover stream. When using stream covers, like a telephone conversation, the sender does not know when the conversation will end. Therefore, he must play safe and make the average distance so short that all the secret bits will be written before the file ends. In some cases, this is not an easy task, because if the distances between the hidden bits are uniformly distributed, then the statistical properties of the noise are not respected (usually the random noise has exponential distribution of interval lengths).

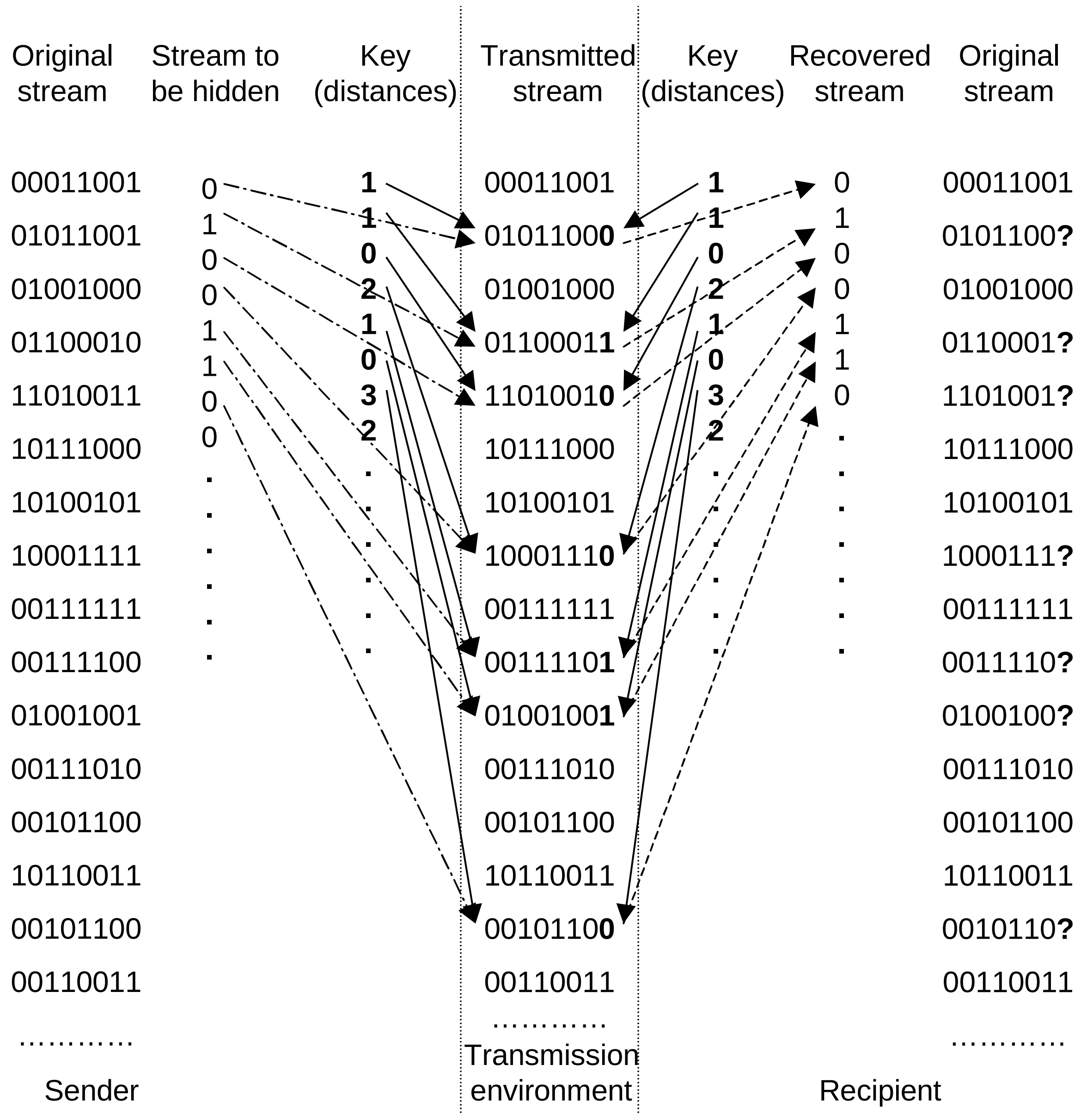


Figure 16. LSB scheme.

Experimental results [ElFra96] showed that if the cover signal is noisy enough, then we can insert a bit every byte, that is approximately 2750 bits per second, but if there is a quiet environment, like when during the phone conversation the two people do not talk, then the distance increases to 15 bytes, that is 180 bits per second. The sampling rate was 28.8 kbps.

3.3.2. Phase Coding

The phase coding method works by substituting the phase of an initial audio segment with a reference phase that represents the secret data. To preserve the relative phase between the segments of the initial file, the phase of following segments is adjusted.

The main benefit of this method is that the signal-to-perceived noise ratio is minimal, so that is one of the most effective coding methods.

When the phase relation between the frequency components changes dramatically, noticeable phase dispersion will be noticeable. But as long as the modifications are sufficiently small the HAS perceives not the modifications.

The general algorithm consists in 6 steps, as described in [WaBen96]:

1. Break the sound sequence $s[i]$, $0 \le i \le I-1$, into a series of N short segments, $s_n[i]$, where $0 \le n \le N-1$
2. Apply a K-points Discrete Fourier Transform (DFT) to nth segment, $s_n[i]$, where $K=I/N$, and create a matrix of the phase ϕ and magnitude A (the dimension of ϕ and A is $K \times N$).
3. Store the phase difference between each adjacent segment for $1 \le n \le N-1$ and $0 \le k \le K-1$: $\Delta\phi[n,k] = \phi[n,k] - \phi[n-1,k]$
4. A binary set of data is represented as a $\phi_{data} = \pi/2$ or $-\pi/2$ representing a "0" or a "1": $\phi'[0,k] = \phi_{data}$, for $0 \le k \le K-1$.
5. Recreate the phase matrix by using the phase difference matrix $\Delta\phi$ and the original phase matrix ϕ: $\phi'[n,k] = \phi'[n-1,k] + \Delta\phi[n,k]$, for $1 \le n \le N-1$ and $0 \le k \le K-1$.
6. Use the modified phase matrix ϕ' and the original magnitude matrix A to reconstruct the sound signal by applying the inverse Fourier transform.

For the decoding process, the synchronization of the sequence is done before the decoding. The length of the segment, the DFT points and the data interval must be known at the receiver. The value of the underlying phase of the first segment is detected as a "0" or a "1", which represents the coded binary string. Since ϕ' is modified, the absolute phases of the following segments are modified respectively. The relative phase difference of each adjacent frame is preserved, because the HAS is sensitive to such modifications.

Using such techniques, the coding channel capacity typically varies between 8 bps and 32 bps. [WaBen96] Having a sound with a little background noise, the data can be embedded only rarely. But if the audio signal has a noisy background, then data may be embedded more often, obtaining a capacity of 32 bps approximately.

3.3.3. Echo Hiding

Echo hiding is another method of embedding data in audio signals. It seeks to do it in a robust way, while not degrading the cover signal. Echo hiding seeks to hide data into sounds with minimal degradation. Minimal degradation means that the changes are imperceivable or simply ignored by the listener, considered environmental distortions. In section 2.4 we described the masking phenomenon.

The process is as follows. [DaGru96] There are some distortions introduced in the cover signal, distortions that are similar to resonance found in a room due to walls, furniture, etc. The difference between the modified sound and the original one is similar to the difference between listening to a compact disc on headphones and listening to it from speakers. In the former case, the sound is not affected by any distortion, but in the latter case we heard the original sound plus echoes caused by the room where we are. By correctly choosing the distortions made to the original signal, we can make them indistinguishable, being comparable to those caused by the room's acoustic. Of course there is a limit for the distortions we can make. Adding extra noise will result in severely degradation of the cover.

Having an audio signal, we can add an echo having a certain delay from the original one. As this delay decreases, the two signals blend. At a certain point the HAS perceives not an echoed signal, but a single (distorted) signal. This point is really hard to determine, because it depends on the quality of the recording, the type of the sound, the listener. But in general we may consider that this point occurs around one thousandth of a second.

The encoder uses two delays, one for embedding a "0" and one for embedding a "1". Both delays are below the threshold that the HAS can perceive them as two different signals. If we want more security, we may also chose the amplitude of the echo below the threshold of the HAS.

The general scheme is depicted in Figure 17.

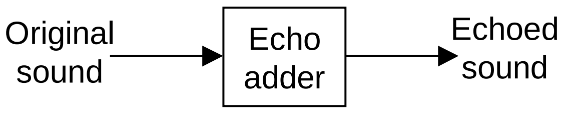


Figure 17. The echoing technique.

The echo adder uses two kernels: the zero and the one kernel, shown in Figure 18.

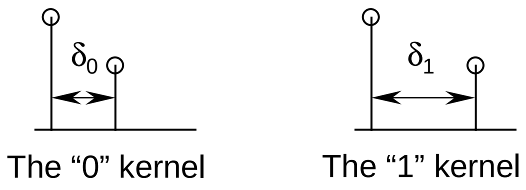


Figure 18. The echo kernels.

The parameters that may be modified are the delay δ, the amplitude of the echo and the delay between the two delays (δ_1-δ_0). By correctly choosing those parameters we can obtain a imperceptible echo added to the signal.

The delay between the cover audio and the echo is chosen accordingly to the value of the bit wanting to embed. If we want to hide a "0" then the delay between the cover and the echo will be δ_0, otherwise the delay will be δ_1(see Figure 19).

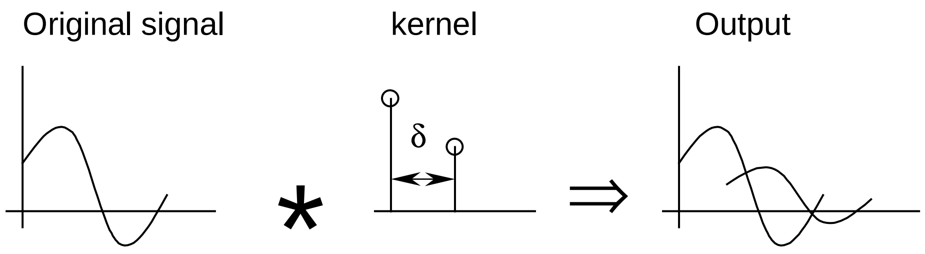


Figure 19. An echoing example.

For encoding more than one bit, the cover is divided into small portions, the number of those portions being equal to the number of the bits embedded. Each portion will be echoed as an independent signal. The output will be the sum, the recombination of all those fragments (and the echoes, too).

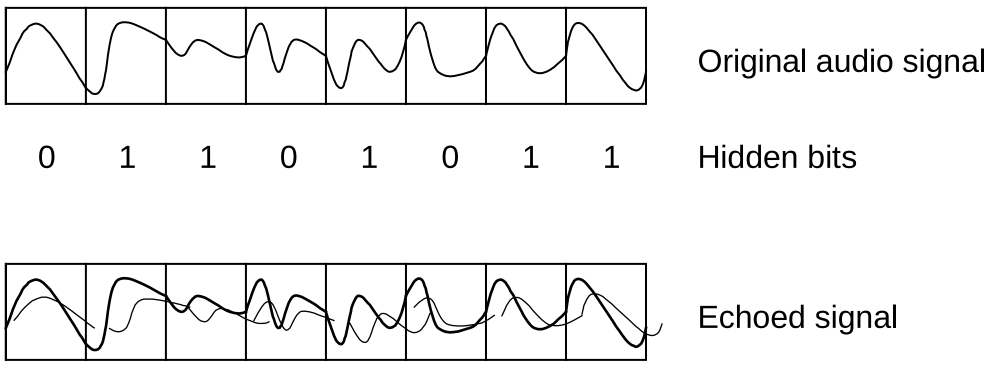


Figure 20. Echoed signal.

The echoed signal from Figure 20 is obtained using the method described above. It contains some abrupt portions, because of the difference between the two delays. Those abrupt portions may be eliminated using a special device or a low pass filter. To simplify the process of echoing, we make a few changes to the scheme. Firstly, there obtain two signals: the first one is the echoed signal using the delay δ_0 and the second is the same signal but having the delay δ_1. Then the two signals are used as inputs for a one-of-two mixer, as shown in Figure 21. The selection signals for the "1" and "0" are inverses, their sum being always unity. This gives a smooth transition between portions encoded with different bits and prevents abrupt changes in the resulting signal, which would be noticeable.

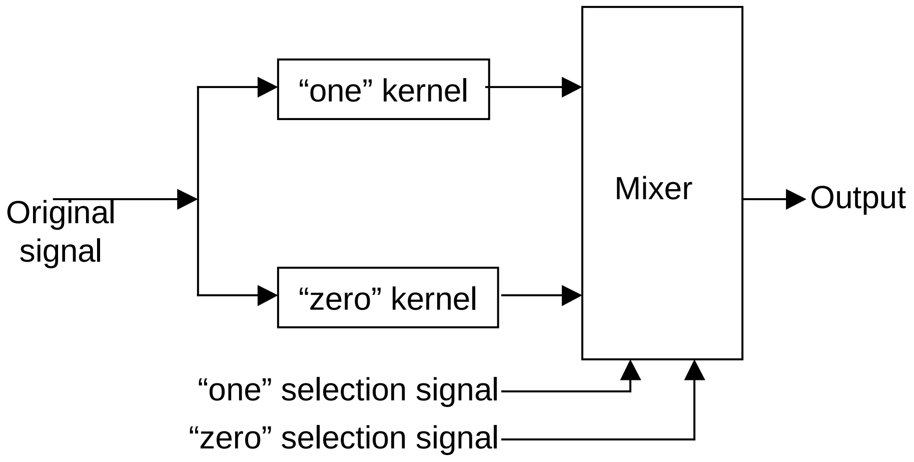


Figure 21. The encoding process.

We saw the manner in which data is embedded into the audio signal, using two different delays for the echoes. The process of extracting the embedded data involves the detection of spacing between the echoes. In order to do this, we will examine the magnitude of the autocorrelation of the encoded signal's cepstrum.

The first step is to transform the encoded signal from *time domain* to *frequency domain*. This can be done using the Fourier transform. A general scheme for generating the cepstrum is depicted in Figure 22.

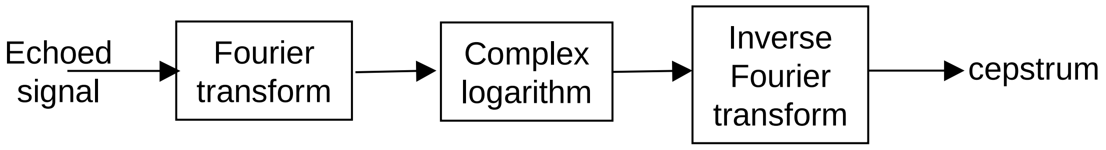


Figure 22. A scheme for generating the cepstrum of a signal.

The operational conversion is the result of a basic mathematical property: the logarithm of a product is the sum of the individual logarithms and multiplication in the frequency domain is identical to convolution in the time domain. We can express this process using mathematical formulas. The echoed signal is $y[n]$ obtained from two signals: $x_1[n]$ and $x_2[n]$:

$$y[n] = x_1[n] * x_2[n]$$

Recall that convolution ($*$) is defined as:

$$x_1[n] * x_2[n] = \sum_{k=-\infty}^{\infty} x_1[k] x_2[n-k]$$

The $x_1[n]$ is the echo introduced and the $x_2[n]$ is the original signal. The echo x_1 is:

$$x_1[n] = \begin{cases} 1, & n = 0, \delta \\ 0, & otherwise \end{cases}$$

By transforming the data from the time domain to frequency domain, using the Fourier transform, we obtain:

$$Y\left(e^{j\Omega}\right) = X_1\left(e^{j\Omega}\right) X_2\left(e^{j\Omega}\right)$$

The next step is to take the complex logarithm of the $Y(e^{j\Omega})$ and then to apply the inverse Fourier transform, as shown further on:

$$\log Y\left(e^{j\Omega}\right) = \log\left(X_1\left(e^{j\Omega}\right) X_2\left(e^{j\Omega}\right)\right) = \log X_1\left(e^{j\Omega}\right) + \log X_2\left(e^{j\Omega}\right)$$

$$F^{-1}\left(\log Y\left(e^{j\Omega}\right)\right) = F^{-1}\left(\log X_1\left(e^{j\Omega}\right)\right) + F^{-1}\left(\log X_2\left(e^{j\Omega}\right)\right)$$

As a result, the cepstrum of the signal y[n] becomes:

$$\tilde{y}[n] = \tilde{x}_1[n] + \tilde{x}_2[n]$$

where $\tilde{x}[n]$ is the cepstrum of x[n].

Having the echoed audio signal, the goal is to determine the echo's delay. The first step is to find the cepstrum of the echoed version. Taking the cepstrum "separates" the echoes from the original signal. The echoes are located in a periodic fashion dictated by the offset of the given bit. As a result, we know that the echoes are in one of two possible locations. The problem is that the result of the cepstrum also duplicates the echo every δ seconds. And also the magnitude of the impulses representing the echoes is small relative to the cover audio. So they are difficult to detect. The solution to this problem is to use the autocorrelation of the cepstrum. A simplified scheme for the autocorrelation of the cepstrum is depicted in Figure 23.

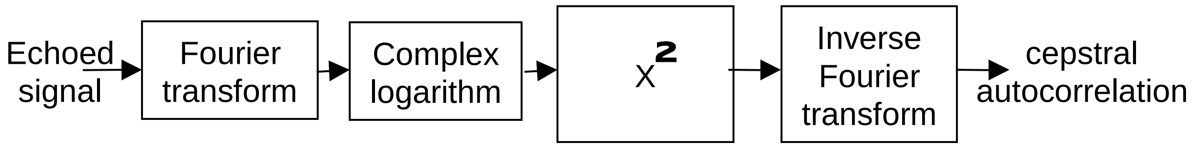


Figure 23. Representation of Cepstral Autocorrelation.

The autocorrelation gives us the power of the signal found at each delay. We will see a "power spike" at either δ_0 or δ_1. Therefore the decision rule is to examine the power at δ_0 and δ_1 and choose the one having the higher power level.

Using the methods described, we can encode and decode information in the form of binary digits in an audio stream with minimal degradation. By minimal degradation we mean that the modifications done to the audio signal are not perceived by the HAS. In most cases the addition of a resonance gives the signal a slightly richer sound.

Experimental results shown that delays greater than 0.5 milliseconds produce acceptable recovery rates. The average listener cannot resolve the echoes with a delay of 0.001 seconds. Below 0.5 millisecond the decoder had difficulty distinguishing the echo from the cover audio.

The algorithm does not work properly in audio files with lots of silent regions. Here, the information cannot be successfully decoded. In some cases the use of error detection/correction mechanisms is sufficient, but where the silent regions are large, the best technique is to avoid those regions, to skip them.

An attack that can be imagined is to detect the echo and to remove it by simply inverting the addition formula. The problem is to detect the echo, without knowledge of either the original object or the echo parameters. In literature this is known as "blind echo cancellation" and is a very difficult task. An alternative is to add multiple echoes with different amplitudes and different delays in order to make impossible the detection using the autocorrelation of cepstrum method. But those modifications will be perceivable.

We have presented techniques for embedding data in audio signals, in the insignificant regions, but also in significant regions. In the following section we will present techniques for inserting data in images.

3.4. Information Hiding in Images

The cheap price of the image manipulation devices has as result that (high quality) images are more and more present in digital form. Nowadays workstation monitors have a resolution of about 100 dpi (dots per inch), fax quality ranges from 200 to 400 dpi, low-cost laser and inkjet printers produce between 300 and 720 dpi, or maybe 1200 dpi and also the scanners.

Like sounds, images are stored on computers as files. Those files may have different formats, but in general an image is an array of numbers that represent the intensity level of each pixel composing the image. If there is a color image, then there is such an array for each of the three primary colors: red, green and blue. The (colored) image is obtained by superposing those three arrays, each pixel being a sum of those three colors.

Digital images are typically stored in either 24-bit (true color) or 8-bit files (color palette). Having a 24-bit picture, we have a better resolution, the length of the file is bigger and also we have more space for hiding information. The 24-bit images use 3 bytes for representing each pixel (one byte for each color). Those 3 bytes can be represented as hexadecimal, decimal or binary values. Having the sequence FFFFFF that means 100 percent red, 100 percent green and 100 percent blue, its decimal value is 255, 255, 255 and its binary value is 11111111, 11111111, 11111111. Combining those three bytes we obtain the white color. The same combining method is applied to each pixel composing an image. [NeJoh98]

Everybody wants to have pictures on his Web page. This can be done using a scanner and the adequate software. The problem is to do it in an optimal way. The rule of thumb when displaying pictures on the Internet is the less information you can give and still have it look good, the better off you are. There are two popular file formats used on the Web, *GIF* (Graphical Interchange Format) and *JPEG* (Joint Photographic Experts Group), both being very different. The question is which of them is better.

The GIF file format is a lossless file compression format. Information on areas with similar colors is compressed. Some GIF file formats support a feature called *interlacing*. Interlacing saves the file with the odd line information first, then the even line information. The idea is that the users get information about what's coming up, so they do not have to wait until the whole image is loaded. In fact, these graphics do not load any faster, they just seem to do. Another feature is *transparency*. For example, having rounded buttons with different background than the page's background does not look good. But is we have a transparent background for the buttons, the problem is solved.

The JPEG file format is a lossy compression format. It does some mathematics to allow images to be compressed from large sizes to smaller ones. When compressing an image too much, some information is lost.

The idea is that when we have large images, like landscape images, the JPEG file format may be the appropriate format and when we have small images, for example small buttons, the GIF file format is adequate. In the latter case, the JPEG may loose some information like the text displayed on the button's face.

For example, having an 800×600 image, the GIF file size could be approximately 1.44 Mbytes, the capacity of one 3.5'' floppy (the size depends on the picture's content). You can imagine that if someone wants to transfer such an image, the time for doing it is relatively long. By transforming the GIF file to a JPEG file, the user can find a ratio compression so that the size of the new file is smaller but in the same time the quality of the image is (almost) the same.

The owners of digital images that publish their work on the Web might be unwilling to allow the distribution of their work in a networked environment if they are not sure that the copyright law is respected. The digital watermarks can solve this problem. As with document images and sound files, digital images can be watermarked, in a way that they contain information about the author and, in some cases, the buyer.

Image watermarking techniques proposed so far can be divided into two main groups: those that embed the watermark in the *spatial domain* and those that operate in the *frequency domain*. Watermarking in the spatial domain means that the watermark is embedded directly into the image, by modifying the arrays we mentioned before. Operating in the frequency domain means that the image is transformed using for example the Discrete Fourier Transform and some coefficients are modified according to a rule. In both cases the modifications should not distort the image, that is, those modifications should not be visible.

As the HAS (Human Auditory System), the Human Visual System (HVS) has some limitations that we can exploit when we make modifications to the images. Some of the characteristics of the HVS that we can exploit are the masking effect of edges, low sensitivity to small changes in luminance, insensitivity to very low spatial frequencies, such as continuous changes in brightness across an image.

What is specific to digital images is that the hiding techniques can have access to any pixel or block of pixels at random (the digital images are random access covers). But the problem is that the images are expected to be subject to many operations, ranging from simple transformations, like translations, to nonlinear transformations, such as blurring, filtering, lossy compression, printing and rescanning, etc.

The algorithm that embeds data into the image should be robust to such attacks, which may be intentional or unintentional. The data hiding techniques must be resistant to as many of those transformations as possible. Another problem that the watermarking scheme should resolve is the rightful ownership of a watermarked object. A possible attack is that a bad guy downloads a watermarked image and embeds a second watermark, pretending that he was the first who inserted a watermark. This attack is known in literature as the *IBM attack*. [LiQia97] [RaWol97]

Further on we will present some watermarking techniques both in spatial domain and the frequency domain.

3.4.1. A LSB Method Based on a Secret Key

The basic idea is to use a *pseudorandom permutation* of the cover bits. [TuAur96] The data is inserted by modifying the chosen bits. Let *N* be the number of cover bits available and let P_0^N be a permutation of the numbers from *0* to *N*-1. Having a watermark with the length *n*, we can hide this secret message into the cover bits: $P_0^N(0), P_0^N(1), ..., P_0^N(n-1)$. The permutation function must be pseudorandom, it has to select bits in an apparently random order. As a result, the hiding bits will be spread all over the cover.

The permutation function must depend on a secret key *K*. Therefore we need a pseudorandom permutation generator, a function which has as input the key *K* and produces as output different sequences of $\{0,...,N-1\}$. To respect the Kerchoff's Principle, this generator must be secure, that is nobody can guess the cover bits where information is embedded without knowing the secret key *K*.

To build such a generator, a pseudorandom function generator may be used. The latter can be considered as a black box that produces different unpredictable functions for each secret key value. It can be easily constructed from any secure hash function *H* by concatenating the argument *i* ($i = 0,...,n-1$) with the secret key *K*.

$$f_K(i) = H(K \circ i)$$

where $K \circ i$ is the concatenation of the key *K* and the argument *i*. So we obtain a pseudorandom function $f_K(i)$ that depends on the parameter *K*.

We describe a pseudorandom function generator proposed by Luby and Rackoff. [TuAur96] The $a \oplus b$ operation denotes the bit-by-bit exclusive OR of *a* and *b*, the result having the length of *a*. Having the binary argument *i* of length *2l*, we divide it in two parts, *Y* and *X*, of length *l* and the key *K* is divided into four parts, K_1, K_2, K_3 and K_4. The scheme is as follows:

$$Y = Y \oplus f_{K_1}(X)$$
$$X = X \oplus f_{K_2}(Y)$$
$$Y = Y \oplus f_{K3}(X)$$
$$X = X \oplus f_{K_4}(Y)$$
$$\textit{return } Y \circ X$$

Running the algorithm 2^{2l-1} times yields a pseudorandom permutation of $\{0,...,2^{2l-1}\}$. The authors of this scheme show that the permutation is as secure as the pseudorandom function generator.

The following algorithm is based on the scheme above. Having the image *I* with the dimensions *x* and *y*, to get the index of the *i*th hidden bit the sequence is:

$$Y = i \operatorname{div} x$$
$$X = i \bmod x$$
$$Y = \left(Y + f_{K_1}(X)\right) \bmod y$$
$$X = \left(X + f_{K_2}(Y)\right) \bmod x$$
$$Y = \left(Y + f_{K_3}(X)\right) \bmod y$$
$$\textit{return } Y * x + X$$

The returned values *Y* and *X* are the coordinates where the *i*th secret bit will be inserted. $K_1 \circ K_2 \circ K_3 = K$. Having a 800 × 600 image, a secret message of 1 Kbytes and the key $K = 123{,}456{,}789$ then the cover image *N* consists of 480,000 bits and the secret message consists of 1024×8=8192 bits. This means that less than 2 percent of the pixels will be altered. To hide the 500th bit, the algorithm for finding its position is:

$$Y = 500 \operatorname{div} 800 = 0$$
$$X = 500 \bmod 800 = 500$$
$$Y = \left(0 + H(123 \circ 500)\right) \bmod 600 = 7566 \bmod 600 = 366$$
$$X = \left(500 + H(456 \circ 366)\right) \bmod 800 = (500 + 3562) \bmod 800 = 62$$
$$Y = \left(366 + H(789 \circ 62)\right) \bmod 600 = (366 + 1563) \bmod 600 = 129$$

The result is $129 * 800 + 62$, that means that the 500th bit will be embedded into the LSB of the cover pixel whose x-coordinate is 129 and y-coordinate is 62.

The decoding process must compute the same formulas for each bit and read the bit that is at the *X* and *Y* coordinates.

This scheme cannot embed long messages without degrading the picture. For each cover there is a limit of the message size. This depends on the image size, whether or not the image has texture areas.

For an attacker it is impossible to find the right succession without having access to the key *K*. This is because of the last three steps of the algorithm. For being more secure, one can add a fourth level. What an attacker can do is to alter a few random LSB hoping that in this way a part of the message is destroyed. The countermeasures for such an attack are the use of an error detection-correction mechanism. But this will not work in case of lossy compression algorithms.

This method can be combined with the method proposed by Cox et al. [InCox96] This method describes how the cover bits are modified. We extract from a document *D* a sequence of values $V = v_1, v_2, ..., v_n$ into which we insert the secret bits $X = x_1, x_2, ..., x_n$ to obtain an adjusted sequence of values $V' = v_1', v_2', ..., v_n'$. The bits *V* are chosen using the method described above. V' is then inserted back into the document to obtain a watermarked document D'. The attackers can alter D' producing a new document D^*. Given *D* and D^*, a possibly corrupted sequence of values X^* is extracted and compared to *X* for statistical significance.

When we insert *X* into *V* to obtain V' we specify a scaling parameter α which determines the extent to which *X* alters *V*. Three natural formulas for computing V' are:

$$v_i' = v_i + \alpha x_i$$
$$v_i' = v_i(1 + \alpha x_i)$$
$$v_i' = v_i(e^{\alpha x_i})$$

The first equation is always invertible, and the last two equations are invertible only if $v_i \neq 0$, which is generally valid. The first equation may not be appropriate when the v_i values vary widely. If $v_i = 10^6$ then adding 100 may not be sufficient for detecting the secret bit, but if $v_i = 10$, adding 100 will distort the value unacceptably.

We can improve this algorithm by using different coefficients α_i and modifying the above formulas as follows:

$$v_i' = v_i + \alpha_i x_i$$
$$v_i' = v_i(1 + \alpha_i x_i)$$
$$v_i' = v_i(e^{\alpha_i x_i})$$

The α_i coefficients can be viewed as a relative measure of how much we can alter v_i without significant degrading it. Having large α_i means that we can alter v_i by a large factor without degrading the cover.

The decoding process requires the extraction of the possibly corrupted hidden bits and the comparing to the original hidden bits to test their similarity. The measure of the similarity between *X* and X^* is given by the formula:

$$sim(X, X^*) = \frac{X^* \cdot X}{\sqrt{X^* \cdot X^*}}$$

where $A \cdot B = \sum_{i=1}^{n} a_i b_i$, *where* $A = a_1,...,a_n$, $B = b_1,...,b_n$. The authors have demonstrated that if we obtain large values for $sim(X, X^*)$ then *X* and X^* are similar (the document contains *X*).

The choice of *n* dictates the degree to which the hidden bits are spread out among the image and can be adjusted to suit the characteristics of the data.

The experimental results showed that this method could be used for achieving integrity, because in the decoding process we use the *sim* function. Even if the image if cut, a part of the original watermark will be found in the altered image, and if we obtain a value that is superior to a predetermined threshold for the *sim* function, then we can say that the image is signed.

For more robustness, this method can be used with methods for embedding marks in perceptually significant regions of a picture, which we will present in the next subchapters. This ensures that the secret bits embedded remains with the image even after common signal and geometric distortions.

3.4.2. The Patchwork Algorithm

The Patchwork algorithm for embedding data in still images was proposed by Bender, Gruhl, Morimoto and Lu in [WaBen96] and in [DaGru98]. The algorithm is based on a statistical method; it embeds in a host image a specific statistic, one that has a Gaussian distribution. This is usually interpreted as a watermark, in the sense “Is the watermark in the picture, yes or no?” We will present the algorithm further on.

The following simplifying assumptions are made for the analysis presented here. The authors ensure that those assumptions are not limiting. The algorithm operates in a 256 level, linearly quantized system starting at 0; all brightness levels are equally likely; all samples are independent of all other samples.

The algorithm proceeds as follows: take any two points, *A* and *B*, chosen at random in the image. Let *a* equal the brightness at point *A* and *b* the brightness at point *B*; let

$$S = a - b$$

The expected value of *S* is 0 (the average value of *S* after repeating this procedure a large number of times is expected to be 0).

Although the expected value is 0, this does not tell us much about what *S* will be for a specific case. The variance of *S*, σ_S , is a measure of how tightly samples of *S* will cluster around the expected value of 0. *a* and *b* being independent, σ_S^2 can be computed as follows:

$$\sigma_S^2 = \sigma_a^2 + \sigma_b^2$$

Considering that $\sigma_a^2 = \sigma_b^2$ (*a* and *b* are samples from the same set), we have:

$$\sigma_S^2 = 2 \times \frac{(255-0)^2}{12} \approx 10{,}837 \qquad \Rightarrow \qquad \sigma_S \approx 104$$

The standard deviation of *S* is 104. If we repeat this procedure *n* times, letting a_i, b_i and S_i be the values of *a*, *b* and *S* during the *i*th iteration, we can define S_n as follows:

$$S_n = \sum_{i=1}^{n} S_i = \sum_{i=1}^{n} (a_i - b_i)$$

The expected value of S_n is:

$$S_n = n \times S = n \times 0 = 0$$

The variance of S_n is:

$$\sigma_{S_n}^2 = n \times \sigma_S^2$$

The standard deviation of S_n is:

$$\sigma_{S_n} = \sqrt{n} \times \sigma_S \approx \sqrt{n} \times 104$$

We can compute S_{10000} for a picture and if it varies by more than a few standard deviations, we can be fairly certain that this did not happen by chance. In fact, the Patchwork algorithm artificially modifies *S* for a given picture, such that $S_n^{'}$ is many deviations away from expected for a secret sequence of pairs (a_i, b_i).

To encode a picture, there are four steps. The first step requires the use of a specific key with a pseudorandom number generator to choose a pair (a_i, b_i). In the second step, the brightness of the patch a_i is raised by an amount δ (typically in the range of 1 to 5 parts in 256). Next step is to lower the brightness in b_i by the same amount δ. The last step is to repeat the first three steps for *n* (typically *n* is around 10,000).

The decoding process will calculate the $S_n^{'}$:

$$S_n^{'} = \sum_{i=1}^{n} \left[(a_i + \delta) - (b_i - \delta) \right] \qquad \Rightarrow$$

$$S_n^{'} = 2 \times \delta \times n + \sum_{i=1}^{n} (a_i - b_i)$$

That means that each step we accumulate an expectation of $2 \times n$. Thus, after *n* repetitions, $S_n^{'}$ will be

$$\frac{2 \times \delta \times n}{\sigma_{S_n^{'}}} \approx 0.028 \times \delta \times \sqrt{n}$$

As n or δ increases, the distribution of $S_n^{'}$ shifts over to the right.

To improve performance, we can treat patches of several points rather than single points. This way, the noise introduced is shifted to lower frequencies and it is less possible that this noise will be removed by lossy compression.

Concerning the patch shape, we have three possible one-dimensional shapes. If we have small patch with sharp edges, the energy of the patch is concentrated in the high frequency portion of the image spectrum. This way, the distortion is hard to see, but it is very easy to be removed, for example using filtering. Another possibility is to have the patch with smooth edges. In this case, the majority of the information is contained in low-frequency spectrum. The third possibility is to combine the first two, obtaining a patch, which tends to distribute the energy around the entire frequency spectrum. In other words, the first alternative can be used for achieving *protection against detection*, the information being imperceptible, but easy to remove; the second alternative achieve *protection against removal*, the information is hard to remove (see Figure 1).

Another aspect concerning the shapes is their arrangement. This aspect has an impact on the visibility of patches. Three possibilities are shown in Figure 24.

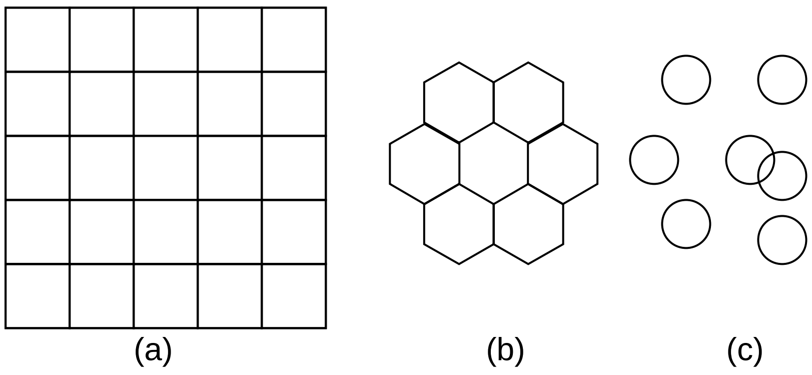


Figure 24. Patches placement.

The first alternative, Figure 24a, is a simple rectilinear lattice. For a big number of shapes, this arrangement is not suitable. Also, the edges formed are visible. Figure 24b shows an alternative to the first method. A preferred solution is in Figure 24c, where the patches are chosen completely randomly. An intelligent distribution of patches in both the horizontal and vertical dimensions will have good results.

The Patchwork algorithm is suitable for applications that embed small quantity of data, such as watermarking. This is because the low embedded data rate of this method.

Further on we will present another method that embeds data in the spatial domain.

3.4.3. Data Hiding using Amplitude Modulation

Using amplitude modulation, single signature bits are embedded by modifying pixel values in the blue channel, which is the one the human eye is least sensitive to. These modifications are either additive or subtractive, depending on the value of the bit and proportional to the luminance. The decoding process does not require the existence of the original unmodified image.

A short description of the method is following. [MaKut97] Let *s* be a single bit to be embed in an image $I = \{R, G, B\}$ and $p = (i, j)$ a pseudorandom position within *I*, generated using a key *K* that the two parts share. The secret bit is embedded at position *p* in the blue channel *B* by modifying the luminance $L = 0.299R + 0.587G + 0.114B$ as:

$$B_{ij} = B_{ij} + (2s-1)L_{ij}q$$

where *q* is a constant determining the strength of the signature. The value of *q* is selected depending upon the purpose of the data hiding. By varying the value of *q*, we can achieve either *protection against detection* or *protection against removal* (see Figure 1).

This was the encoding process. In order to recover the secret bit, a prediction of the original value of the cover bit is needed. This prediction is based on a linear combination of the pixels in a neighborhood around *p*. The authors ensures us that cross-shaped neighborhood gives best performance.

The predicted value $\hat{B}_{ij}$ is computed as:

$$\hat{B}_{ij} = \frac{1}{4c}\left(\sum_{k=-c}^{c} B_{i+k,j} + \sum_{k=-c}^{c} B_{i,j+k} - 2B_{ij}\right)$$

where *c* is the size of the cross-shaped neighborhood. In Figure 25 we outlined this technique. The value $\hat{B}_{ij}$ is the mean value of the hachured bits.

To retrieve the secret bit, the difference δ between the prediction and the actual value of the pixel is taken:

$$\delta = B_{ij} - \hat{B}_{ij}$$

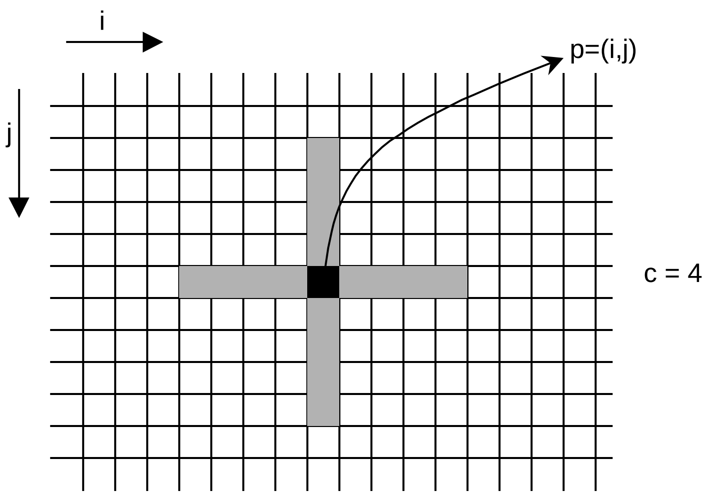


Figure 25. The cross-shaped neighborhood.

The sign of the difference δ determines the value of the secret bit.

The embedding and the retrieval functions are not symmetric, because the retrieval function is not the inverse of the embedding function. The correct retrieval is not guaranteed. To further reduce the probability of incorrect retrieval, the bit can be embedded several times.

Further on we will describe a method for multiple embedding. [MaKut97] The bit can be embedded *n* times at different locations. The *n* positions can be chosen using a pseudorandom number generator, as the one described in section 3.4.1.

We can define the density parameter ρ, which gives the probability of any single bit for being used for embedding. The values of this parameter lies between 0 and 1, where 0 means that no information is embedded and 1 means that information is embedded in every pixel. The number of pixels used for embedding is equal to ρ times the total number of pixels in the image.

The locations for embedding are determined as follows: for each pixel of the image a pseudorandom number *x* is generated. If *x* is smaller than ρ then information is embedded into the pixel, otherwise the pixel is unchanged.

To make the process image size independent, we modify the scanning order. Instead of scanning the image line-by-line, column-by-column, a zig-zag path is taken, as shown in Figure 26.

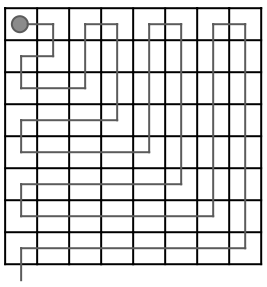


Figure 26. Zig-zag scanning path.

To retrieve the secret bit, the difference between the prediction and the actual value of the pixel is computed for each p_k:

$$\delta_k = B_{p_k} - \hat{B}_{p_k}$$

These differences are then averaged:

$$\bar{\delta} = \frac{1}{\rho|I|}\sum_k \delta_k$$

where $|I|$ is the number of pixels contained in image *I*. The sign of the average difference determines the value of the multiple embedded bit.

If we want to embed an *m*-bit signature $S = \{s_0, s_1, ..., s_m\}$, then let $p_1, ..., p_n$ be the *n* positions selected for the multiple embedded of a single bit. Given an *m*-2 bit string to be embedded, 2 bits are added to the string to form an *m*-bit signature. Those 2 bits are 0 and 1. The reason to do so is that we can define a threshold that improves the signature retrieval and the resistance to geometrical attacks is increased. The threshold can be defined as the average between $\bar{\delta}^0$ and $\bar{\delta}^1$. The decision is as follows:

$$\bar{s} = \begin{cases} 1, & if \bar{\delta} > \dfrac{\bar{\delta}^0 + \bar{\delta}^1}{2} \\ 0, & otherwise \end{cases}$$

The experiments showed [MaKut97] that this method embeds data that is resistant to common attacks, such as: blurring, JPEG encoding, rotation, even to cropping and composition with another image.

3.4.4. A Superposition Algorithm

Further on we will describe an algorithm which embeds data in still pictures using superposition. [IoPit95]

Having an image of dimensions $N \times M$ we can consider a representation of this image as follows:

$$I = \left\{ x_{nm}, n \in \{0,...,N-1\}, m \in \{0,...,M-1\} \right\}$$

where $x_{nm} \in \{0,1,...,L-1\}$ is the intensity level of pixel (n,m). The secret bits can be viewed as a binary pattern of the same size where the number of “ones” equals the number of “zeros”:

$$S = \left\{ s_{nm}, n \in \{0,...,N-1\}, m \in \{0,...,M-1\}, s_{nm} \in \{0,1\} \right\}$$

Using *S* we can split *I* into two subsets of equal size:

$$\begin{aligned} &A = \left\{ x_{nm} \in I, s_{nm} = 1 \right\} \\ &B = \left\{ x_{nm} \in I, s_{nm} = 0 \right\} \\ &|A| = |B| = \frac{|I|}{2} = \frac{N \times M}{2} = P \\ &I = A \bigcup B \end{aligned}$$

The watermark is superimposed on the image as follows:

$$C = \left\{ x_{nm} \otimes k, x_{nm} \in A \right\}$$

where $\otimes$ is the superposition law. The watermarked image is given by:

$$I_S = C \bigcup B$$

The signature is a two-dimensional signal f_{nm}:

$$f_{nm} = \begin{cases} k, & s_{nm} = 1 \\ 0, & s_{nm} = 0 \end{cases}$$

Let $\overline{a}, \overline{b}, \overline{c}$ and s_a, s_b, s_c be the sample mean values and the sample standard deviations of the pixels belonging to subsets *A*, *B* and *C*. The process of detecting the watermark is based on the examination of the difference $\overline{w}$ of the mean values $\overline{c}$ and $\overline{b}$:

$$\overline{w} = \overline{c} - \overline{b}$$

If the watermark is present then $\overline{w}$ is close to *k* whereas in the case of an unmarked image or an image with a watermark different from the one that we are looking for $\overline{w}$ is approximately 0. In other words, $\overline{w}$ is a random variable whose mean is zero for an unmarked image and *k* for an image that contains the watermark.

We can use also a statistic test for indicating the presence of the watermark. Let *q* be:

$$q = \frac{\overline{w}}{\hat{\sigma}_{\overline{w}}}$$

$$\hat{\sigma}_{\overline{w}} = \frac{s_c^2 + s_b^2}{P}$$

There are two alternatives: there is **no** signature in the image or there **is** a signature on the image. In order to decide whether the image is watermarked or not, the value of *q* is tested against a threshold *t*. If *q*>*t* we assume that the image is marked, otherwise we conclude that the image bears no watermark. The threshold value must ensure that the errors are minimal. There are two possible errors. The first error is to decide that the image is watermarked although there is no watermark inside the picture and the second type of errors is to decide that we accept that there is no mark inside the picture although there is one. The value of the threshold is given by:

$$t = \frac{k}{2\hat{\sigma}_{\overline{w}}}$$

In Figure 27 we showed the two possible types of errors.

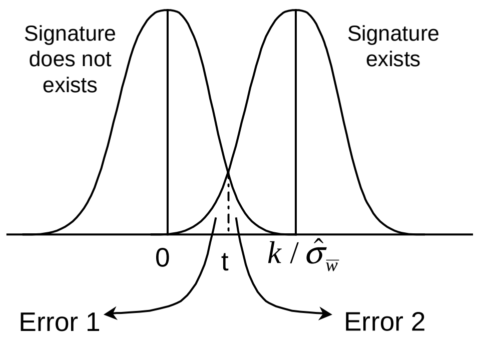


Figure 27. The two types of errors.

We obtain the value for k:

$$k = 2\hat{\sigma}_{\overline{w}} t$$

Resuming, the encoding process is to generate *S*, decide for the desired certainty level (decide if the mark should be resistant against detection of removal), calculate the value of *k* and cast the watermark. The decoding process needs to calculate the *S*, evaluate *q* and compare it to *t* to decide if the watermark exists.

The above algorithm is not resistant to big compression ratios and low pass filtering. To make it more robust to such attacks there are two possibilities. [IoPit96]

The first way to achieve compression robustness is to use blocks for embedding data, not only one pixel, for example 2×2 or 3×3. Instead of randomly selecting *P* of the image pixels for marking we choose *P*/4 randomly positioned blocks of size 2×2.

The second improvement uses the fact that in order to decide for the existence of a signature the algorithm checks the difference between the **mean** values ($\overline{w} = \overline{c} - \overline{b}$). Therefore, the algorithm will give the same results if, instead of superimposing the same constant *k* to all the pixels in *A*, we will use a different value k_{nm} for each pixel *p*, respecting the condition:

$$\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f_{nm} = Pk$$

One possible scenario is to produce the signature *S*, to calculate the value *k*, to divide the signal f_{nm} into blocks of size 8×8 and then to calculate the optimum value k_{nm} for each block. This way we can achieve more protection against low passing filtering or lossy compression algorithms.

3.4.5. The SSIS Method

The Spread Spectrum Image Steganography (SSIS) method provides the ability to hide and recover, error free, a significant quantity of information bits within digital images, avoiding detection by an observer. [LiMar98] Furthermore, it is a blind scheme because the original image is not needed to extract the secret information. The only thing that the recipient must have in order to reveal the secret message is a key.

The SSIS method combines techniques from spread spectrum communication, error-control coding, image processing. The fundamental concept is that the data is embedded in the noise, which is added to the original image. Because the noise is low power and the decoding process is not perfect, a low-bit error-correcting code is incorporated. The main blocks of the encoding process are shown in Figure 28. The message is optionally encrypted with key1 and then encoded via a low-rate error-correcting code, producing the encoded message *m*. The sender also provides key2 for generating a spreading sequence *n*. The modulation scheme is used to spread the narrowband spectrum of the message *m* with the spreading sequence, this way composing the embedded signal *s* which is input into an interleaver and spatial spreader using the key3. The signal obtained is added to the original image *f* to produce the stegoimage *g*, which transmitted in some manner to the recipient.

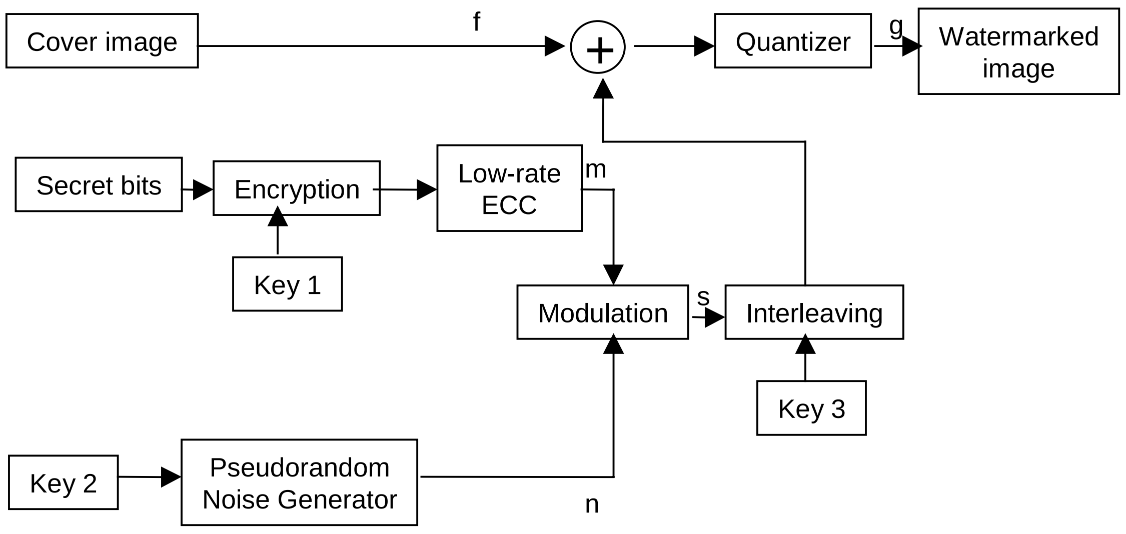


Figure 28. SSIS Encoder.

The receiver has the same keys and the stego image and uses a decoder in order to read the watermark. As shown in Figure 29, the decoder uses image restoration techniques to produce an estimate of the original cover image, $\hat{f}$, from the received image $\hat{g}$. The difference between $\hat{g}$ and $\hat{f}$ is fed into a keyed deinterleaver to construct an estimate of the embedded signal $\hat{s}$. With the key2 the spreading sequence n is reconstructed and then using demodulation an estimate of the encoded message $\hat{m}$ is constructed, which is then decoded using a low-rate error-control decoder.

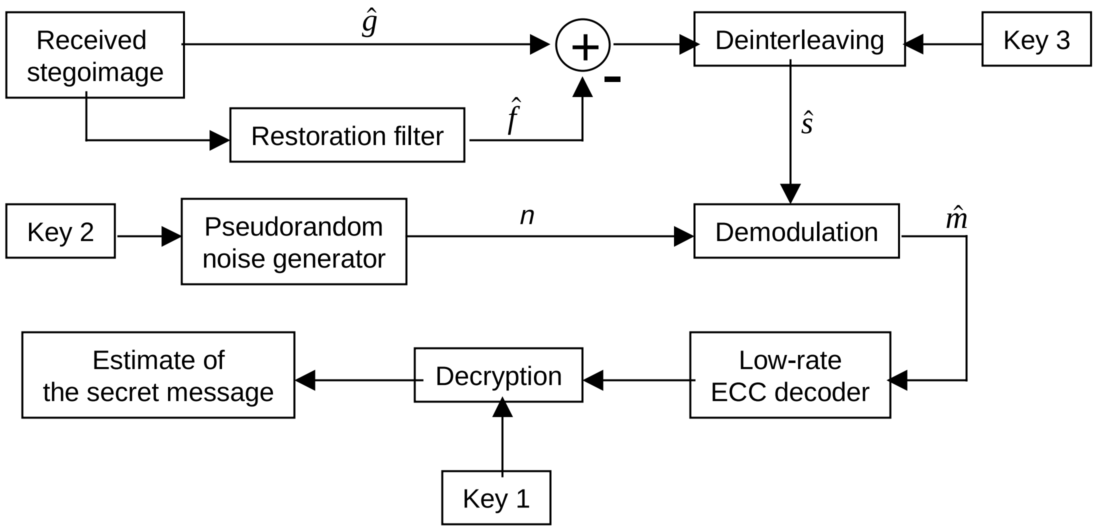


Figure 29. SSIS Decoder

SSIS method uses the concept of spread spectrum communications to embed a message, which is a binary signal, within very low power noise. The resulting signal, perceived as noise, is then added to the cover image to produce the stegoimage. Since the power of the embedded signal is low compared to the power of the cover image, the signal-to-noise ratio is also low, thereby indicating lower perceptibility by the HVS. An observer will be unable to distinguish the original image from the stegoimage with the embedded signal.

A different technique that embeds data in frequency domain is presented further on.

3.4.6. Embedding Data in Frequency Domain

A watermarking technique that operates in frequency domain contains a step in which the image is transformed. Image transformation is done usually using the *Discrete Cosine Transform* (DCT). There are algorithms that use another transforms, such as the Walsh Transform or Wavelet Transform. [MaBar97] The transformation can be applied to the image as a whole or to its subparts. The watermark casting is done by modifying some coefficients, which are selected according to a watermarking rule.

The coefficients to be modified are chosen according to the type of protection needed: if we want our watermark to be imperceivable then the high range of frequency spectrum is chosen, but if we want our watermark to be robust then the low range of frequency spectrum is selected. Usually, the coefficients to be modified belong to the medium range of frequency spectrum, so that a tradeoff between perceptual invisibility and robustness is achieved.

A description of a method for embedding data in frequency domain is presented. [MaBar97]

Let the watermark be $X = \{x_1, x_2, ..., x_M\}$ where *M* is the watermark size. The technique used for embedding data is to superimpose the above sequence to some coefficients of the DCT computed on whole image.

First, the *N*×*N* DCT of the image *I* is computed. The first $L+M$ DCT coefficients are selected to generate a vector $T = \{t_1, t_2, ..., t_{L+M}\}$. In order to obtain the perceptual invisibility without loss of robustness, as discussed above, the first *L* coefficients are skipped. Then the watermark $X = \{x_1, x_2, ..., x_M\}$ is embedded in the last *M* numbers. In this way we obtain a new vector, $T' = \{t_1', t_2', ..., t_{L+M}'\}$ is obtained, using the following formula:

$$t_i' = \begin{cases} t_i, & i = 1,..,L \\ t_i + \alpha \cdot t_i \cdot x_i, & i = L+1,..,L+M \end{cases}$$

where α is a coefficient that is chosen accordingly to the desired level of robustness. If α is increased then the robustness is enhanced.

For decoding, we have a possibly watermarked image I^*. We compute the *N*×*N* DCT of the image and the first $L+M$ DCT coefficients are selected to generate a vector $T^* = \{t_1^*, t_2^*, ..., t_{L+M}^*\}$. As in the encoding process, the first *L* coefficients are skipped and the next *M* are used for calculating the correlation with each of the possible watermarks (we suppose that an author has information about the watermarks he embedded in his documents). The correlation is computed as follows:

$$z = \sum_{i=1}^{M} t_{L+i}^* x_i^*$$

The sequence producing the highest value of *z* is considered the embedded watermark.

An improvement can be done in order to enhance the robustness of the watermark. We can adapt the formula used for embedding the watermark by exploiting the visual masking properties of the HVS. The original image *I* and the watermarked image *I'* are added pixel by pixel, according to a local weight factor $\beta_{i,j}$, obtaining a new image *I"*:

$$y''_{i,j} = y_{i,j}\left(1-\beta_{i,j}\right)+\beta_{i,j}y'_{i,j}$$

where $y_{i,j}$ is the pixel from the (i,j) coordinates in *I* and $y'_{i,j}$ is the similar pixel in *I'*. In regions with low noise sensitivity (for example high textured regions) $\beta_{i,j} \approx 1$, therefore $y''_{i,j} \approx y'_{i,j}$, whereas in regions sensitive to noise (for example uniform regions) $\beta_{i,j} \approx 0$, do $y''_{i,j} \approx y_{i,j}$, the watermark being embedded with a minor extent.

Combining this last improvement with an adequate choice of α, we obtain a robust technique for embedding data in frequency domain.

The experiments [MaBar97] showed that this scheme is robust to JPEG coding, low pass and median filter, blurring, sharpening, random and uniform noise, resizing the image and also to forgery attack. Watermarking in frequency domain is proved to be one of the best methods for achieving protection against watermark removal.

3.5. Marking Binary Files

One of the principal characteristics of the PC is that is an open architecture. Both hardware and software can be accessed for observation and modification. In fact, this openness has lead to the PC's market success. A drawback is that the PC is a fundamentally insecure platform.

The binary files contain code, which is run by the computer. Unlike all the previous file types (document images, sounds and images), binary files do not contain noise or other properties we can modify. If we add something in a binary file or remove something, the execution will be different or the program will crash.

Software developers have tried to find methods for protecting their work. If the files are distributed using CD-ROMs or floppies then the physical support can contain information, like a serial number or the author's logo. But what happens if the software is sold through the Internet? We have the same problem as with other file types, so we have to find a method for embedding copyright information inside the file.

Another difference between binary files and the other file types is the possible attacks. We do not have geometrical distortions, lossy compressions, etc. Binaries can be observed and manipulated. These malicious activities can be classified into three categories, based on the origin of threat. [DaAuc96] These three categories are:

- *Category I*: the malicious threat originates outside of the PC. The perpetrator must breach communication access controls, but still operates respecting the communication protocols. This is the standard *hacker attack*.
- *Category II*: the perpetrator has been able to introduce malicious code into the platform and the operating system has executed it. It must still utilize the operating system and BIOS interfaces. This is a common *virus*.

- *Category III*: the perpetrator has complete control of the platform and may substitute hardware or system software and may observe any communication channel. In this case the owner of the system is the perpetrator.

Category I attacks require correctly designed and implemented protocols and proper administration. Category II attacks are caused by the introduction of a malicious software into the system. The malicious software has been introduced with or without the user's consent. Category III attacks are impossible to prevent on the PC. What we can do is to rise a technological bar to a high sufficient to deter a perpetrator by providing a poor return on their investment. The technological bar would be:

- *no special tools required* (only standard debuggers and system diagnostic tools)
- *specialized software analysis tools* (specialized debuggers and breakpoint-based analysis)
- *specialized hardware analysis tools* (processor emulators and bus logic analyzers)

Watermarking schemes for binary files should be resistant against category II and III attacks.

3.5.1. A Simple Method for Watermarking

We propose a method that embeds a watermark in binary files. We suppose that there exists a tool that is able to find in a given portion of a source code the instructions that can be switched. For example having the following sequence:

```
a = 2;
b = 3;
c = b+3;
d = b + c;
```

this is equivalent to:

```
b = 3;            b = 3;            b = 3;
a = 2;            c = b+3;          c = b+3;
c = b+3;          a = 2;            d = b + c;
d = b + c;        d = b + c         a = 2;
```

The initializations of `b, c and d` must be done in the same order, but `a` can be initialized at any time.

Having a given source code, a method for embedding a watermark $W = \{w_1, w_2, ..., w_N\}, w_i \in \{0,1\}$ will split the source code into *N* blocks and for each block *i* the tool will switch the instruction accordingly to the secret bit w_i. If $w_i = 0$ then the code is left unchanged, otherwise two instructions are switched.

The decoding process requires the original binary file. By comparing the watermarked and the unmarked file we can find the sequence *W*.

This method is very simple, but it is not resistant to attacks. If an attacker has access to many watermarked versions of the same binary, he can easily detect the unmarked version and also he can remove the watermark.

To improve the above scheme, we can modify the encoding technique. If we want to encode a 1 then the number of changes made to the instructions must be odd, otherwise the number of instructions should be even. This way, the attackers cannot guess how the correct source code looks like.

Another possibility is to use computer viruses, which have embedded some public/private key pairs and checks for the integrity of a specific program and also check if the program is being traced. In this case, when something goes wrong, a special jump function can be used, which continues the execution in a location where the program crashes or will run forever.

Better results are obtained if we use more complicated software with special protocols, as described further on.

3.5.2. Tamper Resistant Software

In this subchapter we will describe another possibility for watermarking executable files, a method proposed by David Aucsmith that is resistant to tampering. [DaAuc96]

Tamper resistant software should be immune from observation and modification. This requirement implies that the software contains a secret component. This secret component compels the user to use that specific software for that specific function rather than other software. For example, consider the need to guarantee that the software has completed a predetermined set of steps. If each step contributed some information to the formation of a shared secret then the presentation of the secret would provide proof that the steps have been executed correctly.

The design principles for the tamper resistant software are:

- *disperse secrets in both time and space* – the secret should never exist in a single memory structure, where it could be retrieved by scanning the active memory and also the secret should be processed in more than one operation
- *obfuscation of interleaved operations* – the complete task to be performed by the software should be interleaved in a way that each little part is performed in successive iterations of the executing code. Also, the actual execution should be obfuscated to prevent easy discovery of the interleaved component results. Such obfuscation could be accomplished by self-decrypting and self-modifying code
- *installation unique code* – each instance of the software should contain unique elements, watermarks. This uniqueness can be achieved by adding at program installation different code sequences or encryption keys
- *interlocking trust* – the correct performance of a code sequence should be mutually dependent of the correct performance of many other code sequences.

None of these principles alone will guarantee tamper resistance; it has to be built from many applications of those ideas aggregated into a single software entity.

Tamper resistant software architecture consists of two parts:

1. *Integrity Verification Kernels (IVK).* These kernels have been armored using previously mentioned principles; they can be used alone, to ensure that their tasks are executed correctly, or they can be used together with other software, to provide the assurance that the software has executed correctly
2. *Interlocking Trust Mechanism.* This mechanism uses a robust protocol so that IVKs may check other IVKs, to increase the tamper resistance of the system as a whole.

The Integrity Verification Kernel is a small segment of code which is designed to be included in a larger program and performs two functions: verifies the integrity of code segments of programs and communicates with other IVKs. To accomplish these tasks securely, an IVK utilizes five defenses:

1. *Interleaved tasks*. The functions performed are interleaved so that no function is complete until they are all complete. For example having the tasks *A*, *B* and *C*, where *a*, *b* and *c* are small parts of these, the IVK executes abcabcabc rather aaabbbccc.
2. *Distributed secrets*. The IVK contains a secret; in general this is a private key used to generate digital signatures. The public key would be used to verify the integrity of the program.
3. *Obfuscated code*. The IVK is encrypted and self-modifying so that it decrypts in place as it is executed; when one piece of code become decrypted other sections become encrypted and memory locations are used for different pieces of code at different times.
4. *Installation unique modifications*. Each IVK is constructed at installation time in such a way that even for a given program each instance contains different IVKs.
5. *Non-deterministic behavior*. Where possible the IVK utilizes the multithreading capability of the platform to generate confusion for the attackers.

The Interlock Trust Mechanism protects the executing programs from manipulation. An IVK is included in every program; each IVK is responsible for the integrity of the program in which is embedded. The integrity is verified using digital signatures. An IVK could verify the integrity of any other software component, in addition to the program in which is contained and also a program may have more than one IVK.

The architecture assumes that there is a System Integrity Program running on the PC that is available to all programs. The System Integrity Program contains a special IVK, called Entry Integrity Verification Kernel (eIVK), which has a public interface that can be called by any IVK using a special protocol.

Each IVK is divided into 2^N cells, where $N > 2$. Each cell contains executable code and, with the exception of first cell, they are encrypted. Cells are executed in a pseudorandom order, determined by a key. After the execution of a cell is finished, there is a special *decrypt and jump* function which is executed. The next cell is chosen and is decrypted. The new cell first process some part of a digital signature, then some additional function, as checking if the program is traced. After that, the *accumulator* function is executed and one value is added in an accumulating product. This accumulating product can be checked to see if all previous steps were executed correctly and in correct order. After the accumulator function has run, the cell is executed and then its decrypt and jump function will be run.

The installation process is very important. At the beginning, one or more public/private key pairs are used to generate code that produces digital signatures. This is the watermark; it is different for every copy of the program. Then this code is added to a prewritten code which contains code for the accumulator function and other code for tamper detection. During the installation, a special tool determines the number of cells to be used, allocates code segments to the cells, adds the accumulator and decrypt and jump functions.

We have described methods for watermarking document images, sound files, images and binary files. In the next chapter we will discuss about the security of these schemes and their implications in our life.

Chapter 4. Considerations about Steganographic Techniques

In the previous chapters we described some steganographic techniques used for embedding data in diverse file types: document images, sounds, images and binary files. These methods are used for embedding copyright information in the file types mentioned above. In this chapter we will discuss the security of these steganographic schemes.

We said in the introduction that steganography is synonymous with establishing covert channels. In our case these covert channels were used by the two prisoners for communicating secretly and by the authors for embedding copyright information. The question is: are these the only applications of steganography? The answer is no, as we will see in this chapter.

4.1. Robustness of Steganographic Schemes

As mentioned in the introduction, a necessary condition for a watermark to be considered secure is that the watermark should respect the Kerchoff's Principle: its security must reside in a key. But in some cases an attacker can create problems without even knowing the key.

As with watermarks used for marking the banknotes, there is not a general reader for watermarks. We cannot build a device which accepts as input a file and produce an output containing the watermark existent in that file. The watermarking techniques can differ, for example the watermarking techniques used for the US dollars are not the same methods used for marking the German Marks. Going further, nobody knows all the watermarks embedded in the US dollars, excepting some people from the national bank. Therefore if somebody wants to build a "general watermark reader", he must consider that there are several techniques used for marking an object and some of these techniques are not even public.

If the goal of a secure cryptographic method is to prevent an interceptor from gaining any information about the plaintext encrypted, the goal of a secure steganographic method is to prevent an observant from even obtaining knowledge of the presence of the secret data. If we combine encryption with steganography we achieve even more security.

Having a watermarked object, which we want to publish on the web, this object follows many ways to all the recipients. A publisher must be aware about the possible distortions that an object can undergo and he must consider that an attacker can also introduce these distortions. A robust steganographic scheme must be resistant to these attacks.

For example, considering the document images, the printers may have different resolutions, resulting in different layouts of the documents. Shrinkage or expansion of copy size is present in almost every reproduction device. We do not have to forget the attackers that could introduce such defects in order to make the embedded code unreadable.

To counter the effect of page size changes, the information is encoded in the relative rather than absolute position of textual objects. The information is also encoded independently along both the width and the length of the page, in order to countermeasure the nonlinear geometric defects.

Image distortions is usually more severe in the case of *paper direction*. [JaBra95] Variable paper thickness, drums and wheels out-of-round, nonconstant paper speed, etc. all contribute to more distortions in the paper direction. Since a recovered document may have been reproduced on different paper directions, it is recommended to use simultaneously both word and line encoding to increase decoding performance.

The same thing is valid for images and it is very easy to imagine similar scenarios for sounds and binary files.

The purpose of a watermark is to protect the owner's copyright. It should be resistant against removal, as shown in Figure 1. This means that the scheme design must be done carefully, because everyone can manipulate the watermarked object and claim that he is its owner. In the literature this is called the "rightful ownership" problem or the "IBM attack" [LiQia97] [RaWol97]

Given a watermarked object, it is possible for an attacker to watermark the watermarked object again, using any watermarking scheme. The resulting watermarked object has both the original and the attacker's watermarks on it. Therefore both the original owner and the attacker can claim the ownership, so we have a problem.

If we use formulas then the above scenario can be denoted as: [LiQia97]

$$V \oplus W \Rightarrow V_W \qquad (1)$$

where V is the original object
W is the author's watermark
V_W is the watermarked object.

This is the watermarking process. The attacker creates a watermark W_F without knowing V, extracts W_F from V_W and creates a counterfeit object V_F. Notice that:

$$V_F \oplus W_F \Rightarrow V_W \qquad (2)$$

This way the attacker claims that the original object is V_F and therefore V_W is his watermarked object.

If (2) can be achieved then that scheme is called *invertible watermarking*. Otherwise it is called *non-invertible watermarking*.

A watermarking scheme is invertible if for any object V_W there is a mapping E^{-1} such that:

$$E^{-1}(V_W) = (V_F, W_F) \quad \text{and}$$
$$E(V_F, W_F) = V_W$$

where V_F is the falsified object
W_F if the attacker's watermark
E is the process of embedding the watermark and
the construction of E^{-1} is computationally feasible.

Otherwise the scheme is non-invertible.

An extreme example of attack is that an attacker can choose one bit i from the watermarked image V_W that is zero and set it to one to form his falsified original V_F and claim that his watermark is $(00....010...00)$ where only the i^{th} bit is 1. In this simple scenario, who is guilty?

There are two proposals to solve the ownership problem. One solution is to use a one-way function to map the *N* bits of the original image *V* to a bit sequence $b_i(i = 1,...,N)$ and then to use the bit sequence to arbitrate between two watermarking schemes. Foe example, if we use the pseudorandom generator described in section 3.4.1, if $b_i = 0$ then we use formula $v_i^{'} = v_i + \alpha x_i$ and if $b_i = 1$ then we use the formula $v_i^{'} = v_i(1 + \alpha x_i)$ for embedding the watermark, where v_i are the original bits, $v_i^{'}$ are the watermarked bits, x_i are the secret bits and α is the strength coefficient.

The second proposal is to use schemes that need the original unmarked object for extracting the watermark. The schemes that do not require the original object are invertible, because there is no way to verify how the original object looks like. Because of this, any kind of watermarks and originals are legitimate, therefore an attacker can use a scenario as described above to claim his ownership.

Another proposal is to have only one company that embeds watermarks that can be used in court. This company can also use secure timestamps for resolving the IBM attack. But there is very difficult if not impossible to do this.

4.2. Establishing Big Brother Using Steganography

As discussed at the beginning of this chapter, steganography can be considered a technique for establishing covert channels. There are a lot of other projects that use covert channels. For example there is MIT Media Lab project called *Things that Think.* [YvDes96] The idea is to put sensors and microcomputers in objects, clothes, for example belt buckles, tie clasps. These chips would communicate among themselves and with sensors. The purpose of this system is foe example to allow a user to be identified when arriving to a hotel and then the elevator knows which floor to take him too, the door to his room opens when he approaches, etc.

Supposing that an agency wants to find out, secretly, who buys documents containing maps or a topic considered of interest to national security. If the owner of the document asks for the name and the credit card number of the buyer for using them as a key in the watermarking algorithm then it is very easy to make a database with all the clients.

If we extend this scheme to more complicated schemes, like Things That Think described above, we can imagine that the covert channels can easily be used for other purposes than their original. Because a covert channel appears in mediums where two peoples want to communicate secretly, we can easily imagine a covert channel inside an existing covert channel. For example the sensors and microcomputers that open the door at the hotel can also be used for tracing the person that carries them.

We conclude by saying that using modern technology in an ubiquitous way may pose danger to society, in particular it makes Orwell’s 1984 Big Brother scenario technologically feasible in the next century.

The idea is that steganographic techniques have very useful applications, but it could be also dangerous if such covert channel techniques will invade our world. To countermeasure this, the authors should mark the watermarked objects in a visible way, so that a possible buyer be aware of this. In this case we have no guarantee that an unmarked object does not use steganographic schemes.

What we wanted to say in this chapter is that the development of covert channel techniques to protect the right of individuals such as copyright can swerve and can be used against individuals.

References

(in alphabetical order)

[AnTan96] Andrew S. Tanenbaum, "*Computer Networks*", Third Edition, Prentice Hall, 1996.

[BiPfi96] Birgit Pfitzmann, "*Information Hiding Terminology*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[ChCac98] Christian Cachin, "*An Information-Theoretic Model for Steganography*", in *Proceedings of 2nd Workshop on Information Hiding* (D. Aucsmith, editor), Lecture Notes in Computer Science, Springer, 1998.

[ChKau95] Charlie Kaufman, Radia Perlman, Mike Speciner, "*Network Security. Private Communications in a Public World*", Prentice Hall, 1995.

[Alj98] http://www.aljan.com.au/~wchan/stego.htm

[DaAuc96] David Aucsmith, "*Tamper Resistant Software: An Implementation*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[DaGru96] Daniel Gruhl, Anthony Lu, Walter Bender, "*Echo Hiding*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[DaGru98] Daniel Gruhl, Walter Bender, "*Information Hiding to Foil the Casual Counterfeiter*", in *Proceedings of 2nd Workshop on Information Hiding* (D. Aucsmith, editor), Lecture Notes in Computer Science, Springer, 1998.

[DaKah96] David Kahn, "*The History of Steganography*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[ElFra96] Elke Franz, Anja Jerichow, Steffen Moller, Andreas Pfitzmann, Ingo Stierand, "*Computer Based Steganography: How It Works and Why Therefore Any Restrictions on Cryptography Are Nonsense, at Best*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[InCox96] Ingemar J. Cox, Joe Killian, Tom Leinghton, Talal Shamoon, "*A Secure, Robust Watermark for Multimedia*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[IoPit95] Ioannis Pitas, "*Applying Signatures on Digital Images*", *IEEE Workshop on Nonlinear Image and Signal Processing,* Neos Marmaras, Greece, pp. 460-463, June 1995

[IoPit96] Ioannis Pitas, Nikos Nikolaidis, "*Copyright Protection of Images using Robust Digital Signatures*", in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP-96), vol. 4, pp. 2168-2171, May 1996

[JaBra94] Jack Brassil, Steven Low, Nicholas Maxemchuk, Larry O'Gorman, "*Electronic Marking and Identification Techniques to Discourage Document Copying*", *Proceedings of IEEE INFOCOM'94*, vol. 3, Toronto, June 1994.

[JaBra95] Jack Brassil, Steven Low, Nicholas Maxemchuk, Larry O'Gorman, "*Hiding Information in Document Images*", *Proceedings of the 1995 Conference on Information Sciences and Systems*, Johns Hopkins University, 1995.

[LiMar98] Lisa M. Marvel, Charles G. Boncelet, Jr., Charles T. Retter, "*Reliable Blind Information Hiding for Images*", in *Proceedings of 2nd Workshop on Information Hiding* (D. Aucsmith, editor), Lecture Notes in Computer Science, Springer, 1998.

[LiQia97] Lintian Qiao, Klara Nahrstedt, "*Watermarking Schemes and Protocols for Protecting Rightful Ownership and Customer's Rights*", submitted to Academic Press Journal of Visual Communication and Image Representation, November 1997. URL: ftp://a.cs.uiuc.edu/pub/faculty/klara/watermark.ps

[MaBar97] Mauro Barni, F. Bartolini, V. Cappellini, A. Piva, "*Robust Watermarking of Still Images for Copyright Protection*", in *Proceedings of DSP'97, International Conference on Digital Signal Processing,* Santorini, Greece, 2-4 July 1997, pp.499-502

[MaKut97] Martin Kutter, Frederic Jordan, Frank Bossen, "*Digital Signature of Color Images using Amplitude Modulation*", in *Proceedings of SPIE storage and retrieval for image and video databases,* San Jose, USA, February 13-14, 1997

[MaSan95] Maxwell T. Sandford II, Jonatan N. Bradley, Theodore G. Handel, "*The Data Embedding Method*", Los Alamos National Laboratory, 1995.

[NeJoh98] Neil Johnson, "*Steganography*", George Mason University, URL: http://patriot.net/~johnson/html/neil/stegdoc.

[RaWol97] Raymond B. Wolfgang, Edward J. Delp, "*A Watermarking Technique for Digital Imagery: Further Studies*", VIPER Laboratory, Purdue University, USA.

[RoAnd96] Ross Anderson (volume editor), "*Information Hiding*", *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[RoAnd98] Ross Andersen, Fabien A. P. Petitcolas, "*On the limits of Steganography*", to appear in IEEE J-SAC Vol. 16 No. 4, May 1998.

[SuJaj98] Neil F Johnson, Sushil Jajodia, "*Exploring Steganography: Seeing the Unseen*", *Computing Practices*, IEEE, 1998.

[ThHan96] Theodore G. Handel, Maxwell T. Sandford II, "*Hiding Data in the OSI Network Model*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[TuAur96] Tuomas Aura, "*Practical Invisibility in Digital Communication*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.

[WaBen96] Walter Bender, Daniel Gruhl, Norishige Morimoto, A. Lu, "*Techniques for Data Hiding*", IBM Systems Journal, Vol. 35, 1996.

[YvDes96] Yvo Desmedt, "*Establishing Big Brother Using Covert Channels and Other Covert Techniques*", in *Proceedings of the First International Workshop*, Cambridge, UK, May-June 1996, Springer.